Universidad
Carlos III de Madrid
www.uc3m.es

Grado en Ingeniería en Tecnologías de Telecomunicación

Curso académico (2015-2016)

Trabajo Fin de Grado

Estudio y análisis de escenarios de transmisión de datos orientados a Smart Cities

Autor

Juan Alberto Garrido Cortés

Tutor

Daniel Díaz Sánchez

ÍNDICE GENERAL DEL TRABAJO

ÍNDICE DE ILUSTRACIONES

ÍNDICE DE TABLAS

I. INTRODUCTION

I.1. INTRODUCTION TO THE INTERNET OF THINGS

Over the last years we have seen how the Internet usage has exponentially increased, from the presence of traditional desktop computers to the arrival of smartphones and smartwatches.

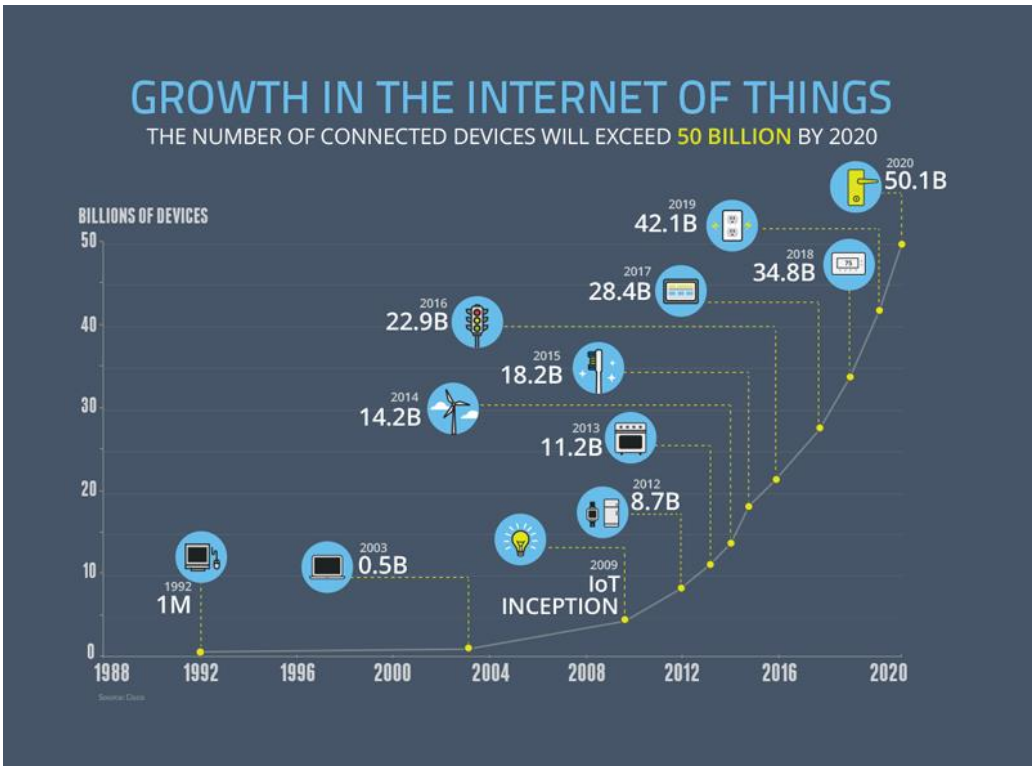

Ilustración 1. Growth Graph Devices Connected to internet. [1]

Now thanks to the Internet, it is easier to see new ways of using technology. This is due to the emergence of "apps", which allow users to perform tasks that they could not have done so easily in the past, as having quick access to their own bank accounts, or even to do bank transfers instantly (something that could take many days before). Another example of new technology is that thanks to the Internet it is now possible to control the video surveillance cameras of a house through mobile phones or tablets.

The current trend is to connect everyday objects to the Internet. In this way, there will come a time in the future when everything will be connected using this technology. This is known as “Internet of Things” (IoT).

This project will analyze the process of interconnection of these objects through internet. It will check how a system in which there are several different objects simulating an environment of IoT works.

I.2. MOTIVATION

Due to the increasing number of Internet-connected devices, new needs appear requiring that the concept of “Internet of Things” can be carried out.

All of those objects connected at the same time, make a network (like the ones we have currently) to be saturated. That is why the need of creating new protocols that improve the performance of a network by making the interconnection of these objects consume little bandwidth (or being able to improve the transmission rate) comes.

What the current technology is seeking for, is to eliminate the presente of a user when handling any type of object or system like the heating thermostat, which necessarily works with human interaction. The aim of all this is to create autonomous systems that can control each other by using sensors and, following with the previous example, to get an auto-adjusting system to always keep the the same temperature in a house.

On the other hand, the sacurity of all that amount of transmissions that will take place in environments where there are many connected devices must be controlled. All the messages containing sensitive information have to be encrypted.

For all these reasons, new protocols will be required in order to accelerate these transmissions, either Machine-to-Machine or Multicast.

I.3. GOALS

The purpose of this paper is to analyse a case study where an environment of Internet of Things with a large number of devices connected to the Internet, is simulated. These devices could be objects of everyday life which can be found in smart-homes (lamps, temperature sensors,...).

It will be studied in more depth how these devices respond by using new Internet communication mechanisms, using new messenger transmission protocols. It is important to perform various tests on different communication options, as it could be the multicast communication or the communication using security mechanisms that allow coded messaging, something that even if it would overload the network, it would also allow a save delivery of sensitive information through it.

This project will try to demonstrate if the use of these new protocols could improve the performance of the devices making them lighter and more efficient, and trying to reduce their energy consumption and lowering their prices.

With the analysis of this simulated environment of IoT it will be discussed, among other things, whether the use of these protocols could eventually replace those we currently use, by exceeding their benefits.

The new protocol that is going to be analysed is CoAP, a messaging protocol to an app level, specialised on the Internet of Things environment, where a large number of devices with limited energy are used. It will try to show that it is a better alternative for the future of this kind of environments.

I.4. CONTENTS OF THE REPORT

In this memory the simulation environment data transmission by using a programming language is going to be studied and analyzed.

As part of a Final Degree Work, the memory will consist of two parts:

- Indexes and sections in mandatory English language: introduction, project summary and conclusion.
- Complete Work content in Spanish: introduction, body of the project and conclusion.

In relation to the Spanish part of the work, it is divided into several different sections:

- Point 1 will begin with the sections of Introduction, Motivation, Objectives and Content of memory.
- Later in point 2 it will continue explaining the state of the art, where it will discuss about the different technologies used in the project.
- Point 3 will be focused on the design and development of the environment, simulating different types of examples.
- Point 4 is the part were we will study and analyzed the previously simulated environments to reach conclusions about the technology.
- After that, an appendix will be included with planning, budgets and the current economic framework.
- In the last section, all links, references and bibliography used during the TFG will be listed.

II. EXTENDED ABSTRACT

The aim of this work is to simulate an environment of the Internet of Things in which there are a large number of devices operating at the same time. These devices would be small everyday objects which would be connected to the Internet and interacting with each other. Examples of these devices could be bulbs, intelligent refrigerators,...

Possibly the most important concept of the project, and also, the one that gives it its meaning, is the concept of Internet of Things (IoT). This concept is something that in a few years will become essential and something that is a real challenge for people having a lot of different connected devices, and in particular, for those responsible of creating the technologies that make it possible.

The Internet of Things is defined as the digital interconnection of objects (especially home devices) to the Internet, so that they can interact with each other without the human intervention to communicate. [2]

Ilustración 2. Example concept Internet of Things. [2]

In the future, the Internet of Things should encode from 50 to 100,000 billions of objects and follow their movement; it is estimated that every human is surrounded by at least 1000 to 5000 objects [4].

Since all those devices will be connected in the future, the need to create new standards and protocols that are able to handle the traffic that they will generate emerge. At the same time, it will be necessary to create new protocols so new devices with performances can be created, making them lighter and with a lower power consumption.

One of these new protocols created to replace the needs that are emerging is CoAP, Constrained Application Protocol. This is a web transfer protocol specialized for its usage with nodes and restricted networks (low power, with losses, ...). These nodes typically have 8-bit microcontrollers with small amounts of RAM and ROM. It is designed to be used machine-to-machine (M2M) in applications such as smart energy and building automation.

CoAP provides a request/response interaction model between the endpoints of the application, that it is compatible with the discovery services and resources, and it includes key Web concepts such as URIs. [9]

It is especially intended to low-power sensors. It is designed to replace the HTTP model but it includes other requirements such as multicast, low overhead and simplicity, which are very important for the Internet of Things (IoT) and Machine-to-Machine (M2M).

CoAP implements the HTTP REST model (with the primitive GET, POST, PUT and DELETE), it uses reduced headers, and limits the exchange of messages, adding UDP support and other modifications like specific security mechanisms [10].

This protocol is fully described in RFC7252.

Right after, some of the most important features of this protocol are going to be exposed:

- Embedded web transport protocol (coap: //).
- The default port for "coap: //" is "5683 / udp" and "coaps: //" is "5684 / udp".
- Ability to use proxies and cache.
- Possibility of using DTLS (Datagram Transport Layer Security) to improve security. There are four levels of security in CoAP: NoSec, PreSharedKey, RawPublicKey and Certificate.
- It withstands multicast and gives the option to create observable resources.
- Possibility to change CoAP traffic to HTTP and vice versa. This is done through proxies called CoAP-to-HTTP proxies.
- Unlike HTTP, CoAP can be ordered asynchronous exchanges on a datagram oriented transport as UDP.

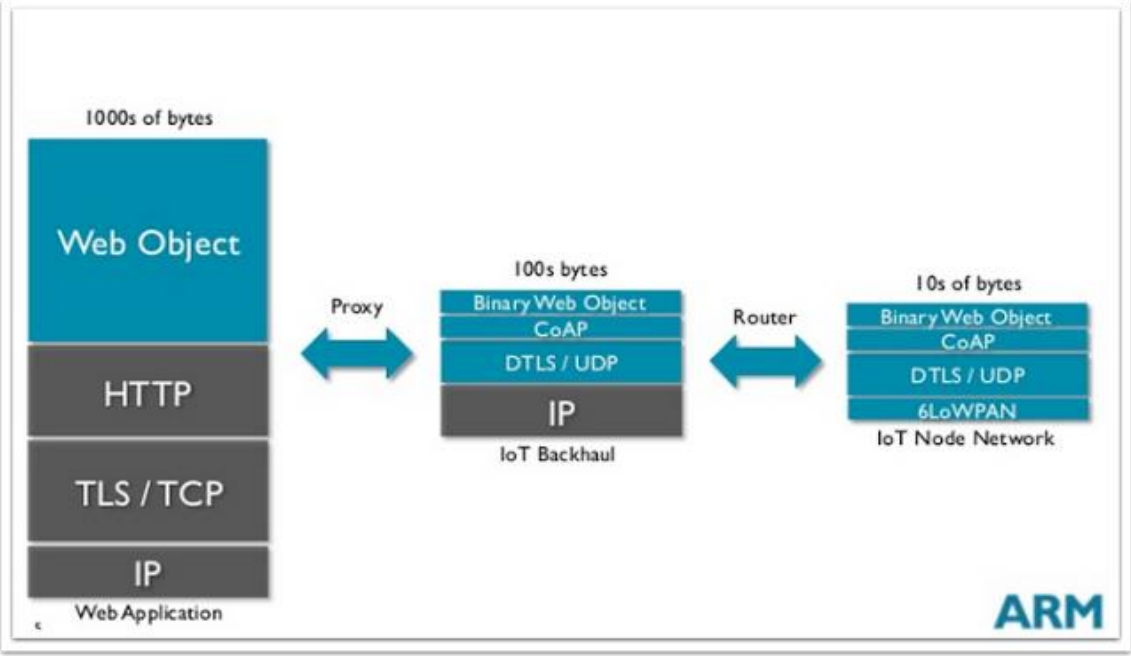

Ilustración 3. HTTP over TCP and CoAP over UDP comparison. [6]

The CoAP messaging model is based on UDP message exchange between its ends.

CoAP uses in its binary header a fixed length of 4 bytes that can be followed by compact binary options and a payload. This applies to both requests and replies.

Next, a basic example of message exchange between a client and a server can be seen:

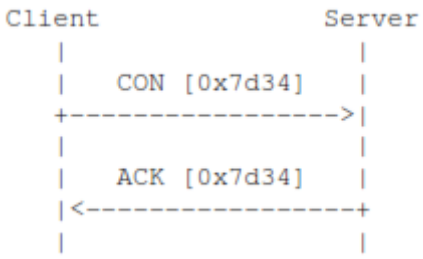

Illustration 4. Example reliable message. [4]

CoAP follows a request / response model, machine to machine. This model meets the following:

- As for the semantics of requests and responses in CoAP messages include a Method Code and Response Code, respectively.
- To mach responses with requests, a Token is used. Note that the Token is not the same as the Message ID.

There are 4 different types of requests in CoAP:

- Confirmable (CON) is used to make requests. It requires an ACK response from the receiver of the request.
- Non-confirmable (NON) is used to make requests and needs no response, ie no need ACK.
- Acknowledgement (ACK) is used to answer. It means that the message has arrived and has been interpreted successfully.
- Reset (RST) is used to answer. It will be returned when an incorrect message is received.

	CON	NON	ACK	RST
Request	X	X	-	-
Response	X	X	X	-
Empty	*	-	X	X

Illustration 5. Summary messaging model. [4]

As it has already mentioned above, a major advantage of CoAP is that it can make requests via multicast IP. This allows a single request to be transmitted to a large number of receptors, and

the other way around, a CoAP client can receive at the same time representations of many servers sources using multicast CoAP on IP multicast.

When searching for development software for this protocol, it is possible to find a large number of them made in different programming languages like Java, C, Phython, Android, etc., some of which are specialised in different design tasks.

JAVA	C / C#	Otros lenguajes
Californium	libcoap	Californium (Android)
nCoAP	CoAP.NET	Node-coap (JavaScript)
Leshan	-	txThings (Phython)

Tabla 1. Ejemplo de software de desarrollo.

For this Final Project, the Java Californium software will be used. It has been chosen because of the easy resources that it has like a high-level language, because it is Open Source and for the amount of design options that Californium has, offering several softwares specialised in different CoAP functionalities such as multicast, DTLS for CoAP,...

Californium (Cf) is an open source implementation of CoAP belonging to Eclipse. It is implemented in Java and contains an API RESTful with all the features of the CoAP protocol.

Californium provides the user with an intuitive and easy-to-use framework to interact with the endpoints of CoAP or provide specific services. In this way, the user doesn't have to deal with the internal parts like retransmissions of messages and block transfers.

The configuration of a CoAP server using Californium, requires nothing more than defining the resources that are going to form it, providing subclasses that implement the handles of GET, POST, PUT and / or DELETE requests. That is, it is not necessary to create from scratch a message. However, the properties of the protocol could allow the user to customize a message thus adapting it to its needs.

The project is divided into 5 sub-projects: Californium (Cf), Scandium (Sc), Actinium (Ac), tools CoAP and Connector.

Once all the technologies that have been used in this project are explained, we will proceed to explain how it has been carried out its design and development.

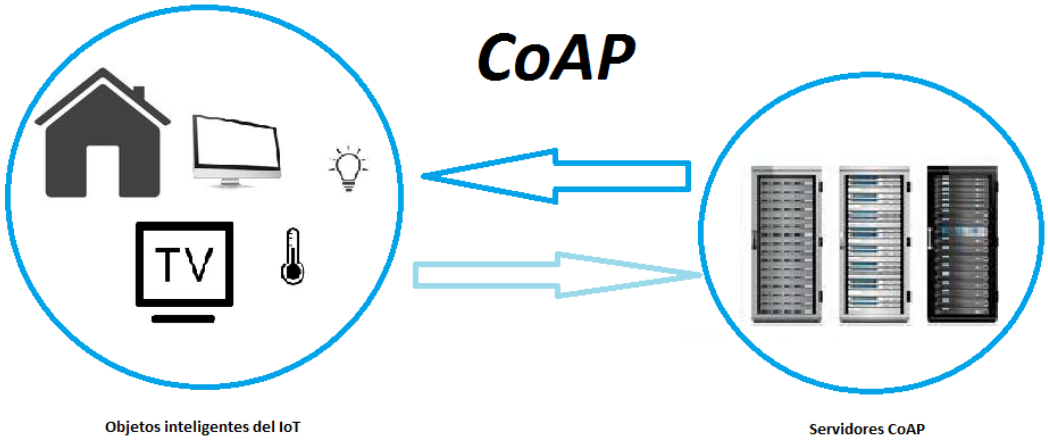

Illustration 6. CoAP simulation example.

- Client. They pretend to be intelligent objects of Internet of Things. Such as a temperature sensor constantly sending data or a bulb.
- CoAP Server. This element is responsible for managing the requests made by clients. Once they receive them, these are processed and, if they require a reply, the server will send a response to the client. This case would be for example when the request is CON, that means it requires an ACK response.

Regarding the development of the solution, this work has used Windows 8.1 operating system with Java 1.8.0_77 version installed. As a developing environment the "Mars.2 Release (4.5.2)" version of Eclipse (Eclipse Java EE IDE for Web Developers) has been used.

In relation to the libraries that are required to develop the project, it has been decided to be imported using the Maven tool. The main library that was needed for the development of this work is version 1.1.0 of Californium (Cf) core.

Regarding the class structure of the project, it was created in Eclipse. It has also been separated into packets, so that in each package an environment is being simulated, testing a specific feature of CoAP Californium.

In relation to the class structure of the project created in Eclipse, it has been separated into packets, so that in each package an environment simulation is going to be made, testing a specific feature of CoAP Californium. These packages are:

- californium.coap.example: this packet contains the source code that will stimulate an environment formed by a customer (bulb, sensor…) and a server, which will exchange messages by using the CoAP protocol.
- Californium.scandium.example: in this packet will be simulated an similar environment to the previous one but now using the Californium package called Scandium to provide security by DTLS to the exchange of messages between client and server.

- Californium.observable.example: this last package is responsible for representing the simulation of an environment in which observable resources will be used, ie a CoAP client observes a resource on a server.

Once the tools and technologies used for the design and development of different CoAPenvironments are already explained, using as support the libraries providing by Californium, a study and analysis of its different tested characteristics has been made.

First of all, we analysed the results obtained in the simulations of the environments previously raised displaying the message exchange that has taken place in them, and then, different properties of the employed technologies will be studied and analysed.

As an example a brief summary of the study that has been done on COAP will be carried out.

After running the Main.java class of the CoAP packet, we get to start the simulation. It has to be remembered that this simulation consists on creating an environment formed by a customer (in this case a temperature sensor) and a server, which exchanged messages with each other.

In the next picture the first data obtained in console when simulating this first package by running the main class are shown:

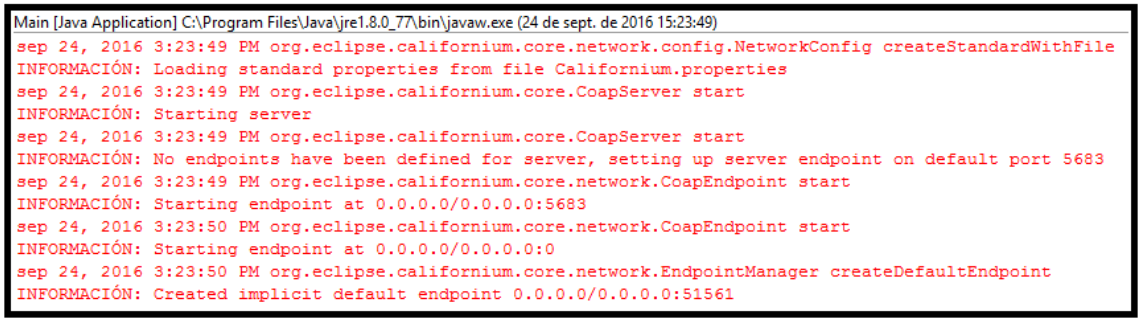

Ilustración 7. Console screenshot - CoAP.

Once the server has started, it will be operational to receive any request from clients or endpoints.

Right after, an image where the results obtained by the customer to make a GET request to the server asking for the temperature is shown:

```
CLIENT: Temperature Sensor
---[Coap Request GET done]---

SERVER
==[ CoAP Response ]=============================================
MID     : 40458
Token   : 624bca6ec26d
Type    : ACK
Status  : 2.05
Options: {"Content-Format":"text/plain"}
Payload: 2 Bytes
----------------------------------------------------------------
24
================================================================
```

Ilustración 8. CoAP GET Request.

It can be observed how the sensor makes the request and get as response a well-defined plot where the different values of the header (MID, Token, Type,...) are perfectly shown and the body, in this case a payload 2 Bytes where the value of the temperature 24ºC is indicated.

At the same time, a comparison between CoAP and other related protocols will be made looking at different characteristics that conect them and, most importantly, that differentiate them.

In the same way, the same studies for CoAP with DTLS and for Observable Resources of CoAP have been made.

III. CONCLUSION

III.1. CONCLUSIONS

The goal of this project was to make several simulations of different environments using the most important features of CoAP. Once these environments have been created the final results have been evaluated and studied.

The motivation of this project came due to the fact that the number of Internet-connected devices has been exponentially increasing during the last years, predicting a future in which all the things that surround us will be connected between them. Thus, the concept of Internet of Things is born and that is why this is the reason for this project.

A protocol capable of facilitate this exchange of messages between devices of everyday life that usually would not count with a lot of power or memory is searched. Thus, we found CoAP, a new protocol specialised in small devices capable of interconnecting two ends.

When going to create environments to exploit the features of CoAP, a development software had to be chosen and, after analysing several of them, Californium was the elected one. This software gave us the tools needed to do the project correctly and to analyse the properties of CoAP by using its libraries.

Once the technologies had been chosen, the greatest difficulty of the project was when the design and the development of environments started. Being a fairly new protocol and not widely used yet there was not too much information, especially practice, that it could help in the implementation of the code.

Finally, we managed to create three project environments: the first one in charge of making a simulation of two ends with CoAP, the second one with CoAP and DTLS, and the third one exploiting the CoAP Observable Resource feature.

Once the environments were created we proceeded to analyse them and we found out that they worked as we had expected they would, and they met the goals set at the beginning of the project. Therefore, it can be considered that the project has been successful because the simulation of an environment of the Internet of Things, and its analysis with the CoAP protocol have been achieved.

In addition, It has been shown that CoAP is clearly a useful protocol for this kind of environments and, probably, it will be one of the most used in a few years.

III.2. FURTHER WORK

With regard to the future work lines on this project it is should be highlighted that there is a large extension margin. This section describes different options that could improve the proposed setting.

A future work line may be creating a complete real environment consisting of multiple devices such as sensors, bulbs and even smartphones. With the creation of this application, the behaviour of this protocol with external factors such as noise could be studied, something that can not be proved in a simulated environment.

Another possible improvement would be the creation of the same simulation set but in this case with another similar protocol like MQTT or HTTP. Thus, a direct and accurate comparison of the performance of these protocols, as well as the speed, energy consumption and code size could be performed.

At the same time, the same environment could be designed, but this time by using another programming language such as C or C#. So it could be checked which one works better. The amount of code, its complexity and how the provided software works could be analysed.

Finally, a new environment in which an example of a CoAP Multicast is simulated, could also be established and studied.

1 INTRODUCCIÓN

1.1 INTRODUCCIÓN

Durante los últimos años hemos podido ver como el uso de Internet se ha visto incrementado exponencialmente, desde la presencia de los ordenadores tradicionales de mesa hasta la llegada de los *smartphones* y *smartwatches*.

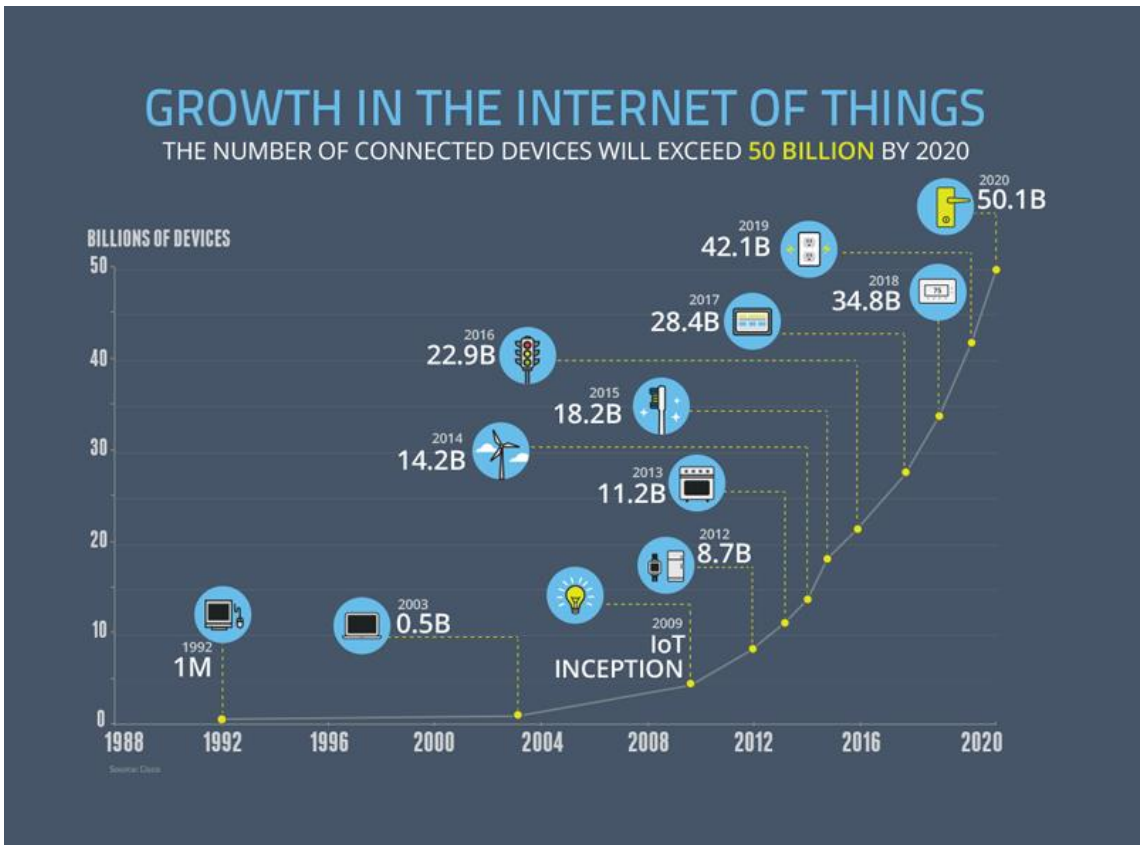

Ilustración 9. Gráfica del crecimiento de dispositivos conectados a internet. [1]

Cada vez se pueden ver más posibles usos de la tecnología a través de internet. Esto se debe en gran medida a la aparición de las "*apps*" las cuales permiten a los usuarios realizar tareas que antes no se podían hacer con tanta facilidad, como el poder acceder de forma rápida a nuestras cuentas bancarias, o incluso realizar traspasos de forma instantánea, algo que antiguamente podría llevar días. Otro ejemplo de nueva tecnología que ha surgido gracias a internet es el poder controlar, a través del móvil o Tablet, las cámaras de video vigilancia de una casa.

La tendencia que se está tomando actualmente es la de conectar todos los objetos cotidianos a internet, de esta forma, llegará un momento en el futuro donde todo esté conectado mediante esta tecnología. Esto último se conoce como **Internet of Things (IoT)**.

En este proyecto se va a analizar el proceso de interconexión de estos objetos a través de internet. Se comprobará como funciona un sistema en el que existan varios objetos distintos simulando un entorno de IoT.

1.2 MOTIVACIÓN

Durante los últimos años se ha ido viendo como cada año aumentaba de forma exponencial el número de dispositivos conectados a Internet, previendo así un futuro en el que todas las cosas que nos rodeen estén conectadas entre sí. De ahí nace el concepto de Internet of Things, el cual es el motivo de este trabajo.

Toda esa cantidad de objetos conectados hacen que una red como las que se tiene actualmente pueda quedar saturada. Por ello surge la necesidad de crear nuevos protocolos que permitan mejorar el rendimiento de una red al hacer que la interconexión de estos objetos consuma poco ancho de banda o consiguiendo mejorar la velocidad de transmisión.

La proliferación de estos pequeños dispositivos ha disparado la creación de nuevos niveles físicos y de enlace centrado en una gestión apropiada de la batería sacrificando ancho de banda (802.15.4 y 6LoWPAN). Por otro lado, la arquitectura de servicios REST ha demostrado ser de gran utilidad a la hora de gestionar grandes cantidades de información en entornos distribuidos. Lamentablemente son muchos los motivos por los que HTTP no es apropiado para estos casos (*overhead*, dificultad de parseado).

Lo que además se busca es eliminar la presencia de un usuario a la hora de manejar algún tipo de objeto o sistema como podría ser el termostato de la calefacción, el cual funciona necesariamente con la interacción humana. El objetivo de todo esto es crear sistemas autónomos que puedan controlarse entre sí por medio de sensores y, siguiendo con el ejemplo anterior, conseguir ajustar la temperatura de una casa siempre a la misma temperatura.

Por otra parte se debe cuidar la seguridad de toda esa cantidad de transmisiones que se llevarán a cabo en entornos en los que haya muchos dispositivos conectados, debiendo ir los mensajes con información sensible cifrados.

Por todo esto se requerirán nuevos protocolos que agilicen estas transmisiones para IoT.

1.3 OBJETIVOS

La finalidad de este trabajo es la de **analizar un caso de estudio en el que se simula un entorno de Internet of Things** donde un gran número de dispositivos están conectados a internet. Estos dispositivos podrían ser objetos de la vida cotidiana como los que se pueden encontrar en casas inteligentes (lámparas, sensores de temperatura,…).

Se estudiará con más profundidad cómo responden estos dispositivos utilizando nuevos mecanismos de comunicación con internet utilizando nuevos protocolos de transmisión de mensajería. Es importante realizar diversas pruebas sobre diferentes opciones de comunicación como podría ser la comunicación multicast o la comunicación empleando mecanismos de seguridad que permitan cifrar la mensajería, algo que a pesar de cargar más la red, permitiría el envío seguro de información más sensible.

Se buscará ver si con la utilización de estos nuevos protocolos se podrían mejorar las prestaciones de los dispositivos haciéndolos más ligeros y eficientes, y que de esta forma pudieran disminuir su consumo de energía y así abaratar los costes que estos supondrían.

Con el análisis de este entorno simulado de IoT se tratará, entre otras cosas, de saber si la utilización de estos protocolos podría en un futuro sustituir a aquellos que se emplean actualmente, ya que estos nuevos superen en prestaciones a los anteriores.

El nuevo protocolo que se va a analizar es **CoAP**, un protocolo de mensajería a nivel de aplicación especializado en entornos de Internet of Things donde se emplea un gran número de dispositivos con energía limitada. Se tratará de demostrar que es una buena alternativa de cara al futuro para este tipo de entornos.

1.4 CONTENIDO DE LA MEMORIA

En esta memoria se va a estudiar y a analizar la simulación de un entorno de transmisión de datos mediante el uso de un lenguaje de programación.

Como parte de un Trabajo de Fin de Grado, la memoria se va a diferenciar en dos partes:

- Índices y secciones obligatorias en idioma ingles: introducción, resumen del proyecto y conclusión.
- Contenido completo del trabajo en idioma español: introducción, cuerpo del proyecto y conclusión.

En cuanto a la parte en español del trabajo, ésta se encuentra dividida en varias secciones diferenciadas:

- Como **punto 1** se iniciará con los apartados de Introducción, Motivación, Objetivos y Contenido de la memoria.
- Posteriormente en el **punto 2** se continuará explicando el Estado del Arte, donde se hablará de las diferentes tecnologías utilizadas en el proyecto.
- El **punto 3** se centrará en el diseño y desarrollo del entorno, simulando diferentes tipos de ejemplos.
- Para el **punto 4** se reserva la parte en la que se estudiarán y analizarán los entornos anteriormente simulado para poder así sacar conclusiones sobre la tecnología.
- Tras esto, se incluirá un apéndice donde se incluirá la planificación, los presupuestos y el marco económico actual.
- Como última sección, se listarán todos los enlaces, referencias y bibliografía utilizados durante el TFG.

2 ESTADO DEL ARTE

2.1 INTRODUCCIÓN

En este punto se van a explicar las diferentes tecnologías que se han empleado en este trabajo, tanto herramientas como conceptos.

Posiblemente el concepto más importante del proyecto y que además es el que le da sentido es el Internet de las cosas (*IoT*). Este concepto es algo que en unos años se volverá imprescindible y algo que supone un auténtico reto para las personas, y en particular, para aquellos encargados de crear las tecnologías que lo hagan posible. Al haber una gran cantidad de diferentes dispositivos conectados, desde una bombilla a un ordenador clásico, existirán un gran número de protocolos, los cuales muchos de ellos solo podrán ser empleados por un tipo de dispositivo.

Un posible problema que puede ocurrir en estos escenario es, por ejemplo, que una bombilla funcione únicamente con la aplicación de su fabricante y con funcionalidades limitadas, de forma que esa misma aplicación no sea funcional con otro tipo de bombilla de otro fabricante o incluso pudiera ocurrir que un modelo más nuevo de la misma tenga otro tipo de aplicación también diferente.

El reto en todo esto es buscar formas de juntar todos estos protocolos y conseguir conectarlos haciendo así un sistema uniforme donde todos los dispositivos se pudieran conectar con todos en caso de requerirlo.

2.2 INTERNET OF THINGS

Ilustración 10. Ejemplo del concepto del Internet de las Cosas. [2]

El **Internet de las Cosas** ("*The Internet of Things*") es definido como la interconexión digital de objetos a Internet, de forma que todos puedan interactuar entre ellos sin necesidad de la intervención humana para comunicarse. [2]

Kevin Ashton, director ejecutivo del Auto-ID Center del MIT (Massachusets Institute of Technology), fue el primero en mencionar este concepto en una presentación que hizo en 1999.

El internet de las cosas debería codificar de 50 a 100.000 billones de objetos y seguir el movimiento de estos; se calcula que todo ser humano está rodeado de por lo menos unos 1000 a 5000 objetos [3]. Según la empresa Gartner, en 2020 habrá en el mundo aproximadamente 26 millones de dispositivos con un sistema de adaptación al internet de las cosas [8].

Pensando entre otras cosas en esta necesidad que iba a surgir se creó el protocolo IPv6, cuyo predecesor es IPv4. Este protocolo permite dar conexión a internet a un mayor número de dispositivos existiendo también la opción de identificar cada uno de los objetos, algo que no se puede hacer con IPv4.

Una de las aplicaciones más importantes donde se utiliza el Internet de las Cosas es en las ***Smart Cities*** (en español, "Ciudades Inteligentes"). Este concepto se refiere a un tipo de desarrollo urbano basado en la sostenibilidad que es capaz de responder adecuadamente a las necesidades básicas de instituciones, empresas, y de los propios habitantes, tanto en el plano económico, como en los aspectos operativos, sociales y ambientales [9]. Estas ciudades tienen seis elementos claves que la conforman: movilidad, medio ambiente, economía, gobernanza, calidad de vida (sanidad, seguridad) y sus habitantes.

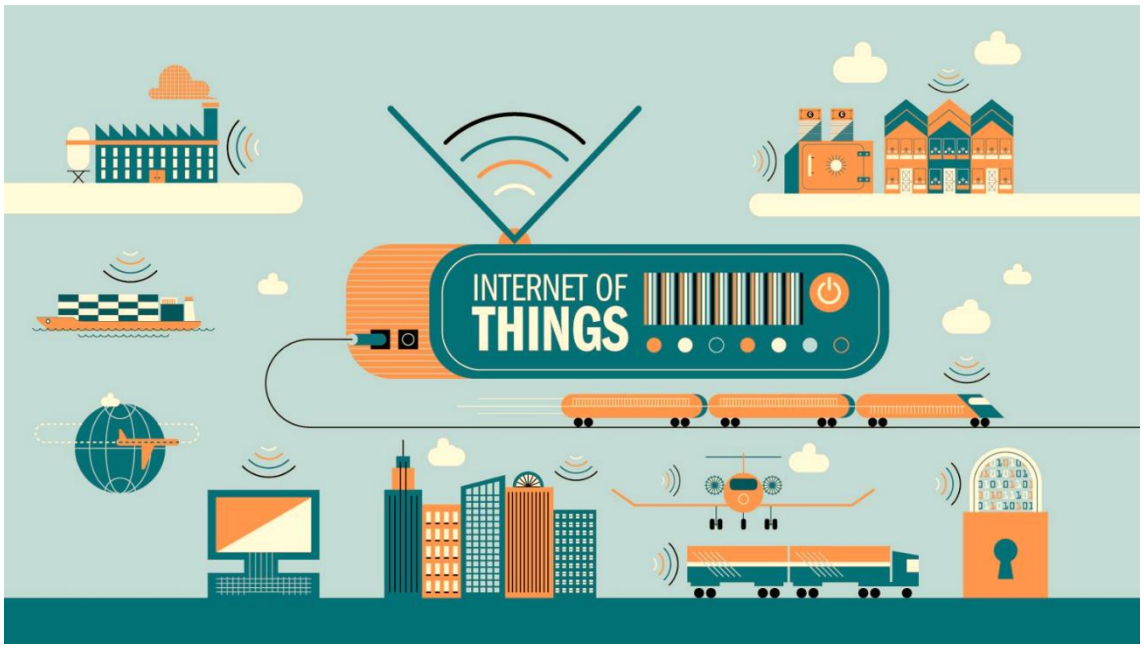

Ilustración 11. Internet of Things en Smart Cities. [8]

Como recoge la Alliance for Internet of Things Innovation (AIOTI) en sus informes, el Internet of Things (IoT) es la base fundamental sobre la cual se construyen las Smart Cities.

Se prevé que en 2050 más del 80% de la población viva en ciudades por lo que éstas deben afrontar un conjunto de retos para la posible convivencia de sus ciudadanos. Algunos ejemplos de estos son el reparto equitativo de bienes y materias primas, el control de la polución, el

abastecimiento energético, un plan de movilidad eficiente para el ciudadano y una optimización de los servicios sanitarios y de seguridad que se prestan a la ciudadanía.

Un ejemplo de IoT en Smart Cities es la **Movilidad Inteligente**, también conocido como "Smart Mobility", donde se incluiría la creación de puntos de carga para vehículos eléctricos.

Otro ejemplo sería la utilización de semáforos inteligentes. Estos semáforos estarían conectados a un circuito de cámaras distribuidas por la ciudad que identifican el nivel de tráfico y de movimiento de masas, evitando las esperas en las zonas de escaso movimiento.

También se pueden aplicar El Internet de las Cosas de forma más profunda al hogar. Distribuyendo una serie de sensores y procesadores, podríamos automatizar el control de las ventanas, la temperatura del hogar, las luces, etc. Y, al estar conectado todo el sistema a internet, también sería posible controlar de forma inalámbrica cualquier objeto.

Ilustración 12. Internet of Things en el hogar. [10]

2.3 CONSTRAINED APPLICATION PROTOCOL (CoAP)

CoAP (Constrained Application Protocol) es in protocolo de transferencia web especializado para el uso con nodos y redes restringidos (de baja potencia, con pérdidas,…). Los nodos por lo general tienen microcontroladores de 8 bits con pequeñas cantidades de RAM y ROM, mientras que las redes restringidas tales como IPv6 sobre 6LoWPANs (Low-Power Wireless Personal Area Networks) a menudo tienen altas tasas de error de paquetes y un rendimiento aproximado de 10 segundos a kbit/s.

El protocolo está diseñado para ser usado machine-to-machine (**M2M**) en aplicaciones como la energía inteligente y la automatización de edificios.

CoAP proporciona un modelo de interacción de **petición/respuesta** entre los extremos finales de la aplicación, compatible con el descubrimiento de servicios (*discovery services*) y recursos, e incluye conceptos claves de la Web tales como URIs. [4]

Está pensado especialmente para sensores de baja potencia. Se ha diseñado para sustituir el modelo HTTP pero incluyendo otros requisitos como multicast, bajo overhead y simplicidad, que son muy importantes para el Internet de las cosas (IoT) y Machine-to-Machine (M2M).

CoAP **implementa el modelo REST de HTTP** (con las primitivas GET, POST, PUT y DELETE), usa cabeceras reducidas, y limita el intercambio de mensajes, añadiendo soporte **UDP** y otras modificaciones como mecanismos de seguridad específicos [11].

Este protocolo aparece descrito en su totalidad en la **RFC7252**.

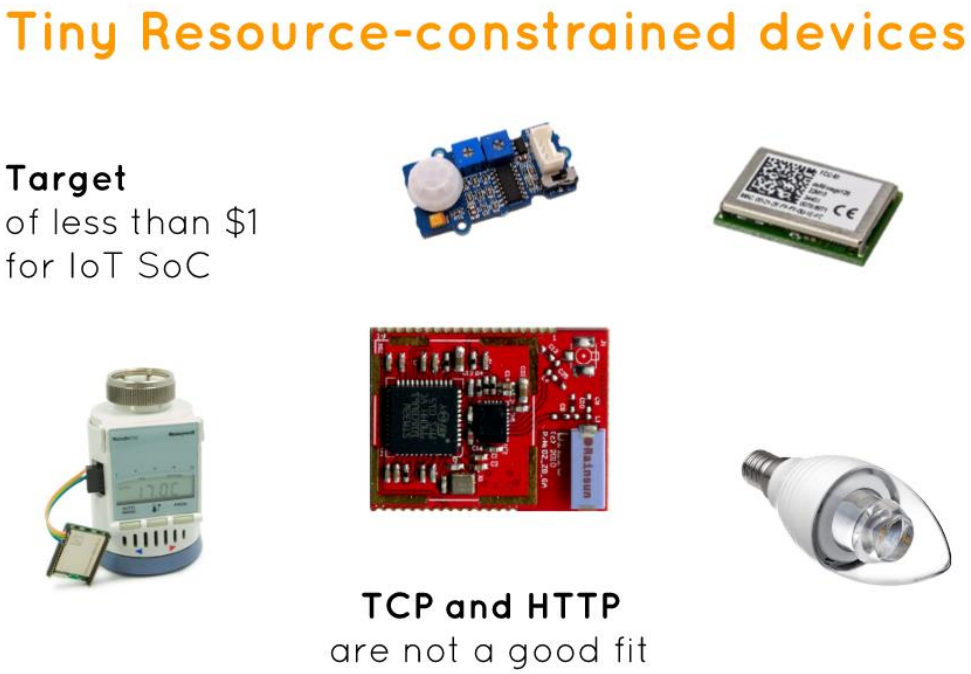

Ilustración 13. Ejemplo de dispositivos que podrían utilizar CoAP. [12]

1) Características

- Protocolo de transporte web embebido (coap://).
- El puerto por defecto para ***"coap://"*** es "5683/udp" y para "***coaps://"*** es "5684/udp".
- Protocolo Web que cumple los requisitos M2M en entornos restringidos.
- CoAP dispone de cuatro tipos de mensajería: Confirmable, Non-Confirmable, Acknowledgement, Reset. Estos serán explicados más adelante.
- Posibilidad de intercambio de mensajes asíncronos.
- Cabecera con un nivel de carga baja y con análisis de complejidad.
- Soporta URIs y Content-type.
- Capacidad de emplear **proxies** y **caché**.
- Posibilidad de emplear **DTLS** (Datagram Transport Layer Security) para mejorar la seguridad.

- Soporta multicast y da opción de crear recursos observables.
- Fácil subscripción para un recurso.
- Posibilidad de pasar tráfico de CoAP a HTTP y viceversa.
- El mapeo de CoAP con HTTP ya está definido, lo que permite a los proxies ser creados proporcionando acceso a los recursos de CoAP a través de HTTP de forma uniforme.
- A diferencia de HTTP, CoAP se puede encargar de intercambios asíncronos sobre un transporte orientado a datagramas como es **UDP**.

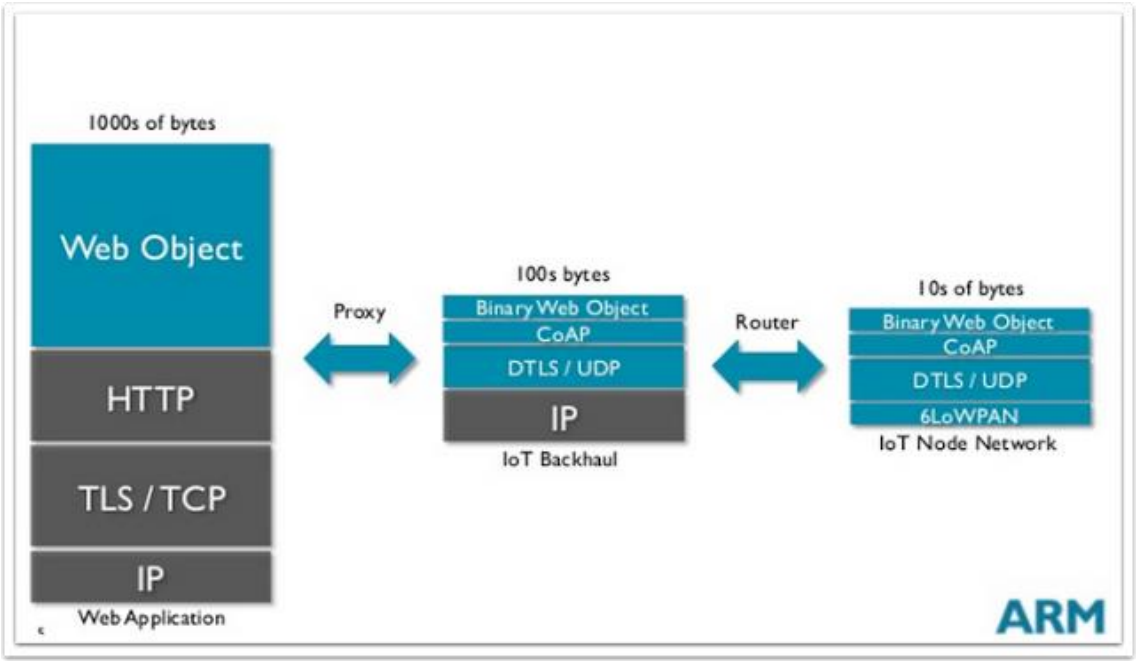

Ilustración 14. Comparación de HTTP sobre TCP y CoAP sobre UDP. [13]

2) Modelo de mensajería

El modelo de mensajería de CoAP está basado en el intercambio de mensajes sobre UDP entre extremos.

CoAP utiliza en su cabecera binaria una longitud fija de 4 bytes que puede ir seguida por unas opciones binarias compactas (*compact binary options*) y un payload. Esto es válido tanto para las peticiones como para las respuestas.

La fiabilidad llega cuando se marca un mensaje como Confirmable (CON). Un mensaje Confirmable es transmitido usando un timeout por defecto y un back-off exponencial entre transmisión, hasta que el receptor envíe un mensaje Acknowledgement (ACK) con el mismo ID de mensaje (en el ejemplo, 0x7d34) que tenía el mensaje enviado por el emisor.

Cuando un receptor no está preparado para procesar un mensaje Confirmable (es decir, ni siquiera estar preparado para enviar una respuesta de error adecuada), este responde con un mensaje RESET (RST) en lugar de un ACK.

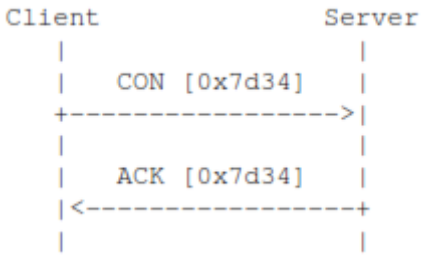

Ilustración 15. Ejemplo de transmisión fiable de mensajes. [4]

Un mensaje que no requiere de fiabilidad es mandado como un mensaje Non-confirmable (NON). A pesar de no haber una respuesta ACK por parte del receptor, sigue habiendo un Mensaje ID para detectar duplicados. Cuando un receptor no está preparado para procesar un mensaje Non-confirmable, este responde con un mensaje RESET (RST).

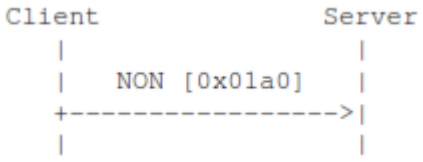

Ilustración 16. Ejemplo de transmisión de mensaje no fiable. [4]

3) Modelo Petición/Respuesta (Request/Response)

Como ya se ha explicado anteriormente, CoAP sigue un modelo de petición/respuesta máquina a máquina. Este modelo cumple lo siguiente:

- En cuanto a la semántica de las peticiones y las respuestas en los mensajes de CoAP se incluye un Method Code y un Response Code, respectivamente.
- Para machear las respuestas con las peticiones se emplea un **Token**. Nótese que el Token no es lo mismo que el **Message ID**.
- Si se envía una petición en un mensaje Confirmable e inmediatamente después se responde a este con un mensaje ACK, esta acción se denomina “piggybacked response”.

A continuación se van a ver dos ejemplos de **piggybacked response** donde se responde a una petición GET, primero de forma exitosa, y después con un error “Not found”.

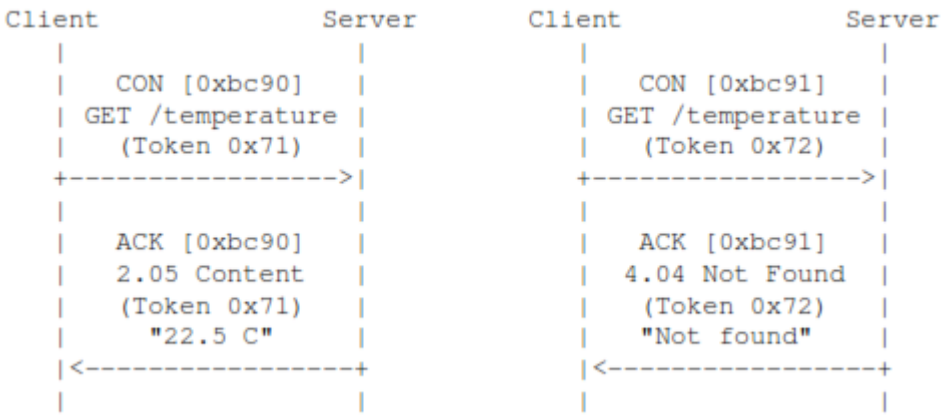

Ilustración 17. Dos ejemplos de piggybacked response. [9]

Si el servidor no puede responder de forma inmediata al mensaje de petición Confirmable, este responde al cliente con una respuesta ACK vacía (Empty Acknowledgement) y el cliente dejaría de retransmitir la petición. Cuando la respuesta esté ya lista, el servidor la enviaría como un mensaje Confirmable (el cual debería ser respondido después por el cliente). Esto es conocido como una "separate response" y se puede ver el proceso en la siguiente ilustración:

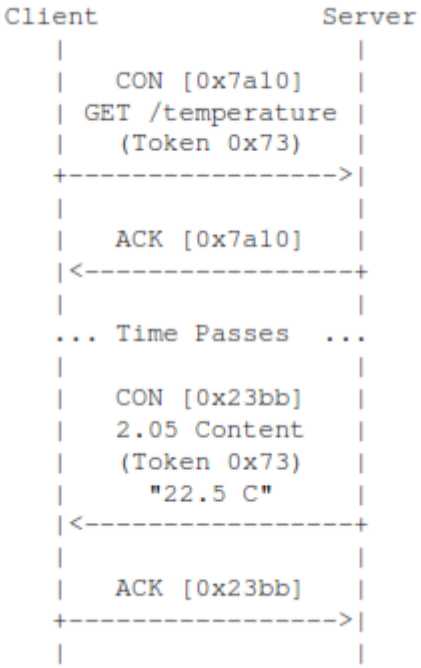

Ilustración 18. Ejemplo de separate response. [4]

Si una petición es enviada en un mensaje Non-confirmable, la respuesta también será enviada con este tipo de mensaje, a menos que el servidor quiera enviarlo como un mensaje Confirmable.

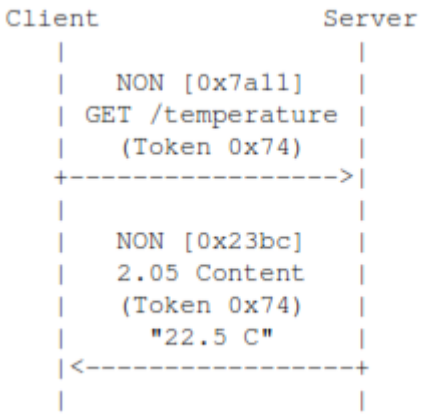

Ilustración 19. Ejemplo de petición y respuesta NON. [4]

CoAP hace uso de los métodos **GET, POST, PUT y DELETE** de forma similar a como lo hace HTTP. Esto se verá más adelante.

4) Formato del mensaje

CoAP está basado en el intercambio de mensajes que, por defecto, son transportados sobre UDP. Además puede ser usado sobre DTLS (Datagram Transport Layer Security) y sobre otros protocolos de transporte como son SMS, TCP, o SCTP.

Cabe recordar que el formato de los mensajes comienza con **4 bytes de cabecera fija**, seguida por la longitud variable del valor del Token, cuya longitud va de 0 a 8 bytes. Después de este valor de Token viene una secuencia de cero o más Options en formato TLV (Type-Length-Value) y de forma opcional un payload que completará el resto del datagrama.

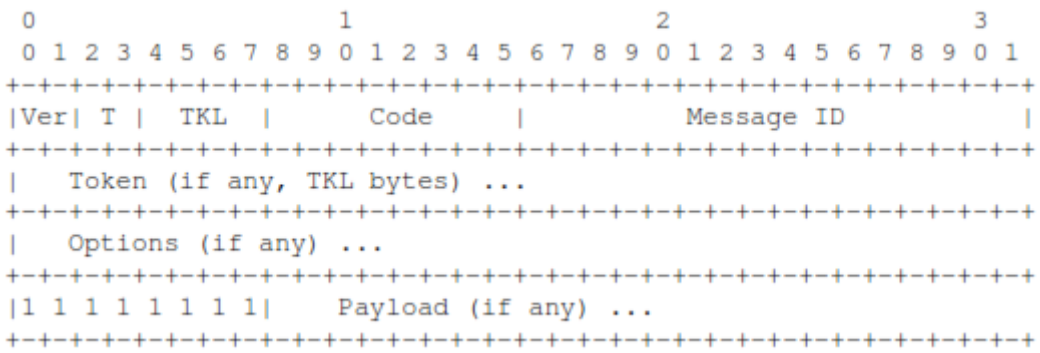

Ilustración 20. Esquema del formato de mensaje. [4]

A continuación se van a explicar brevemente cada uno de los campos que componen el mensaje:

- **Versión** (Ver): 2 bits de unsigned integer. Indica el número de versión de CoAP. Actualmente el único valor que da sentido a este campo es el 1, el resto se reservan para futuras versiones. En caso de recibirse un mensaje con un valor distinto a 1, este sería ignorado.

- **Tipo** (T): 2 bits de unsigned integer. Indica si el mensaje es Confirmable (0), Non-confirmable (1), Acknowledgement (2), o Reset (3). A continuación se explica un breve resumen de la utilidad de cada tipo:
  - **Confirmable** (CON): se emplea para realizar peticiones. Requiere de una respuesta ACK por parte del receptor de la petición.
  - **Non-confirmable** (NON): se emplea para realizar peticiones y no necesita contestación, es decir, no necesita ACK.
  - **Acknowledgement** (ACK): se utiliza para responder. Quiere decir que el mensaje ha llegado y se ha interpretado con éxito.
  - **Reset** (RST): se utiliza para responder. Se devolverá cuando se reciba algún mensaje incorrecto.

	CON	NON	ACK	RST
Request	X	X	-	-
Response	X	X	X	-
Empty	*	-	X	X

Ilustración 21. Resumen modelo de mensajería. [4]

- **Longitud de Token** (TKL): 4 bits de unsigned integer. Este valor marca la longitud del campo Token (0-8 bytes). Las longitudes de 9 a 15 están reservadas y en caso de recibirse se le tratará como un mensaje erróneo.
- **Código** (Code): 8 bits de unsigned integer divididos en la clase, formada por 3 bits (bits más significativos), y el detalle, de 5 bits (bits menos significativos). La clase puede indicar una petición (0), una respuesta exitosa (2), una respuesta errónea del cliente (4), o una respuesta errónea del servidor (5). En caso de estar el campo completo formado íntegramente por ceros, querrá decir que es un mensaje vacío. Si el mensaje es una petición este campo Code se corresponde con el Request Method, y si se tratara de un mensaje, se correspondería con un Response Code.
- **ID del mensaje** (Message ID): 16 bits de unsigned integer. Se utilizan para detectar mensaje duplicados y para que mensajes de tipo ACK/Reset hagan match con mensajes CON/NON.
- **Valor del Token** (Token): Campo opcional. Puede ir de 0 a 8 bytes. Se utiliza para correlacionar las peticiones con las respuestas.
- **Options**: Este campo es opcional y puede desde no existir hasta aparecer varias veces. Cada uno de los campos Options que haya en el mensaje estarán formados por los siguientes sub-campos:
  - Option Delta: 4 bits de unsigned integer. Los valores de 13 a 15, incluidos, están reservados para mensajes especiales.
  - Option Length: 4 bits de unsigned integer. El valor indica en bytes la longitud del subcampo Option Value. Los valores de 13 a 15, incluidos, están reservados para mensajes especiales.
  - Option Value: Algunas de los posibles valores de este campo son: “empty”, “opaque”, ”uint”, “string”.

No.	C	U	N	R	Name	Format	Length	Default
1	x			x	If-Match	opaque	0-8	(none)
3	x	x	-		Uri-Host	string	1-255	(see below)
4				x	ETag	opaque	1-8	(none)
5	x				If-None-Match	empty	0	(none)
7	x	x	-		Uri-Port	uint	0-2	(see below)
8				x	Location-Path	string	0-255	(none)
11	x	x	-	x	Uri-Path	string	0-255	(none)
12					Content-Format	uint	0-2	(none)
14		x	-		Max-Age	uint	0-4	60
15	x	x	-	x	Uri-Query	string	0-255	(none)
17	x				Accept	uint	0-2	(none)
20				x	Location-Query	string	0-255	(none)
35	x	x	-		Proxy-Uri	string	1-1034	(none)
39	x	x	-		Proxy-Scheme	string	1-255	(none)
60			x		Size1	uint	0-4	(none)

C=Critical, U=Unsafe, N=NoCacheKey, R=Repeatable

Ilustración 22. Tabla de posibles Options Value. [13]

- **Payload**: Campo opcional. En caso de existir comenzará por un Payload Marker (0xFF), que indicará el final del campo Options y el comienzo del payload. El final del campo corresponde con el final del datagrama UDP. En caso de no existir un Payload Marker será indicativo de que no existe payload. Si existe Payload Marker pero el resto del payload está formado íntegramente por ceros, se tomará como un mensaje erróneo.

5) Caching

Los extremos en CoAP pueden tienen la opción de cachear las respuestas. Esto podrían hacerlo con el objetivo de reducir el tiempo de respuesta y el ancho de banda de la red. De esta forma podrán reutilizar una respuesta anterior, para responder una nueva pregunta.

A diferencia de HTTP, el cacheo en las respuestas CoAP no depende del método de la pegunta (Request Method), sino del código de la respuesta (Response Code).

6) Proxying

Un proxy es un extremo CoAP que puede ser utilizado por los clientes para realizar peticiones en su nombre. Esto es útil, por ejemplo, para dar una respuesta de una caché y así reducir el tiempo que se tardaría en contestar, el ancho de banda de la red o el consumo de energía. También podría utilizarse para evitar que el extremo realice peticiones que no necesite saber en ese momento, pero que para cuando las necesite las tendrá el proxy y así podría reducir también el tiempo de respuesta.

En la arquitectura general de un entorno RESTful, los proxies pueden tener diferentes propósitos haciendo así varios tipos de proxy:

- Forward proxy: es elegido por el cliente para procesar las peticiones en su lugar.
- Reverse proxy: puede sustituir a uno o varios servidores y procesa las peticiones en su lugar.

- CoAP-to-CoAP proxy: es un proxy que mapea de peticiones CoAP a peticiones CoAP, es decir, tanto el cliente como el servidor utilizan este protocolo.
- Cross proxy: a diferencia del proxy anterior, este transforma el mensaje de un protocolo a otro (CoAP a HTTP y viceversa). Esto es útil, por ejemplo, para pasar tráfico de una pequeña red que utiliza CoAP a una más grande que utilice HTTP.

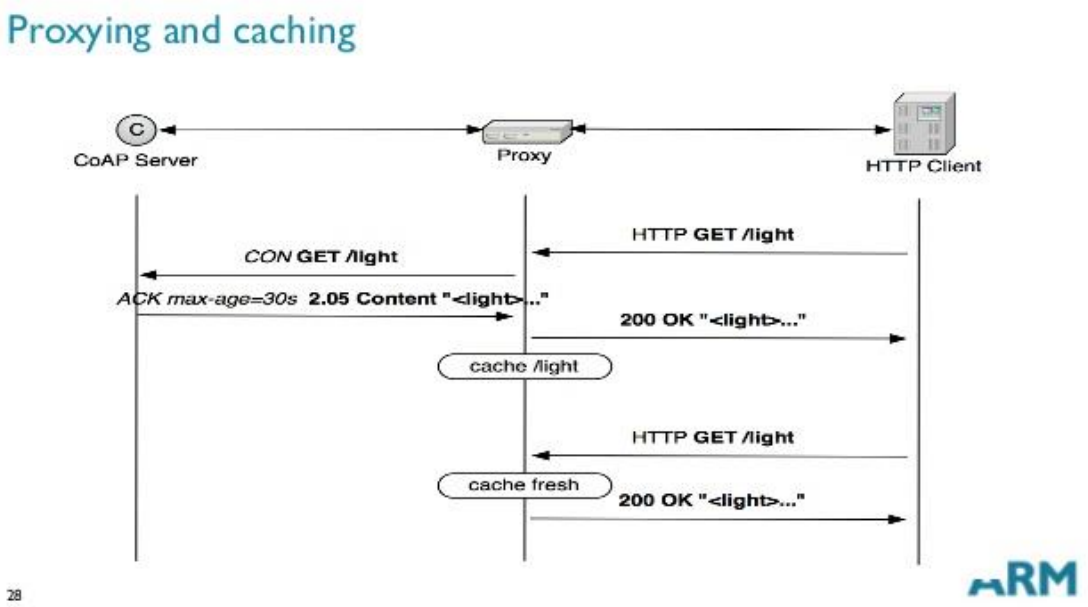

Ilustración 23. Ejemplo de envío de mensajes con Proxying y Caching. [13]

7) Request Method

Una petición con un método de código (Method Code) que no esté reconocido se responderá con un Method Not Allowed (4.05). A continuación se van a definir los diferentes métodos de peticiones de CoAP.

- **GET**: recupera una representación de la información que corresponde a la fuente identificada por el recurso URI.
- **POST**: solicita que la representación incluida en la petición sea procesada.
- **PUT**: solicita que la fuente identificada por el recurso URI sea actualizada o creada con la representación incluida.
- **DELETE**: solicita que la fuente identificada por el recurso URI sea borrada. En caso de éxito se responderá con un Response Code 2.02.

8) Response Code

Como ya se vio anteriormente un Response Code está dividido en el sub-campo Class (3 bits) y el sub-campo Detail.

El apartado Code en el mensaje se documenta de la forma “c.dd” donde “c” corresponde al dígito de la clase (Class) y “dd” corresponde al detalle (Detail) y comprende los números del 00 al 31.

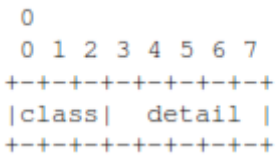

Ilustración 24. Estructura campo Code. [4]

Los 3 tipos de clases que existen son "Success", "Client Error" y "Server Error". A continuación se va a mostrar una tabla donde se muestran los Response Code más utilizados:

SUCCESS (2.xx)	Created	2.01
	Deleted	2.02
	Valid	2.03
	Changed	2.04
	Content	2.05
CLIENT ERROR (4.xx)	Bad Request	4.00
	Unauthorized	4.01
	Bad Option	4.02
	Forbidden	4.03
	Not Found	4.04
	Method Not Allowed	4.05
	Not Accepted	4.06
SERVER ERROR (5.xx)	Internal Server Error	5.00
	Not Implemented	5.01
	Bad Gateway	5.02
	Service Unavailable	5.03

Tabla 2. Ejemplos de Response Code.

9) DTLS en CoAP

CoAP permite mejorar su seguridad utilizando DTLS (**Datagram Transport Layer Security**). Existen cuatro niveles diferentes de seguridad:

- **NoSec**: en este nivel de seguridad la opción DTLS está desactivada por lo que los mensajes se envían en claro. Podrían emplearse otras alternativas de seguridad como es IPsec.

- **PreSharedKey**: en este nivel DTLS está habilitado. Consiste en el empleo de una lista de contraseñas previamente compartidas donde cada una contiene una lista de los nodos que pueden utilizarlas para comunicarse.

- **RawPublicKey**: DTLS está habilitado y cada dispositivo tiene un par de claves asimétricas, una privada y una pública. Este nivel no incluye certificados.

- **Certificate**: contiene lo mismo que el nivel RawPublicKey más el uso de certificados X.509. Para la firma se usará SHA-256.

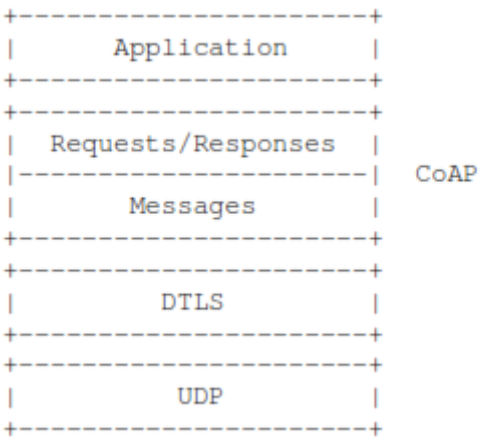

Ilustración 25. Resumen de las capas utilizando DTLS con CoAP. [4]

Una de las cosas que más importa a la hora de implantar seguridad en un protocolo de mensajería es la gran cantidad de tiempo que puede tardar cada paquete en llegar a su destino y el posible aumento de energía que esta capa de seguridad pueda suponer.

Por ello se presenta a continuación dos gráficas las cuales muestran estos dos valores que tanto preocupan. La primera representa la cantidad de energía consumida por el número de Bytes de datos transmitidos y la segunda muestra la relación entre la RTT y el número de Bytes.

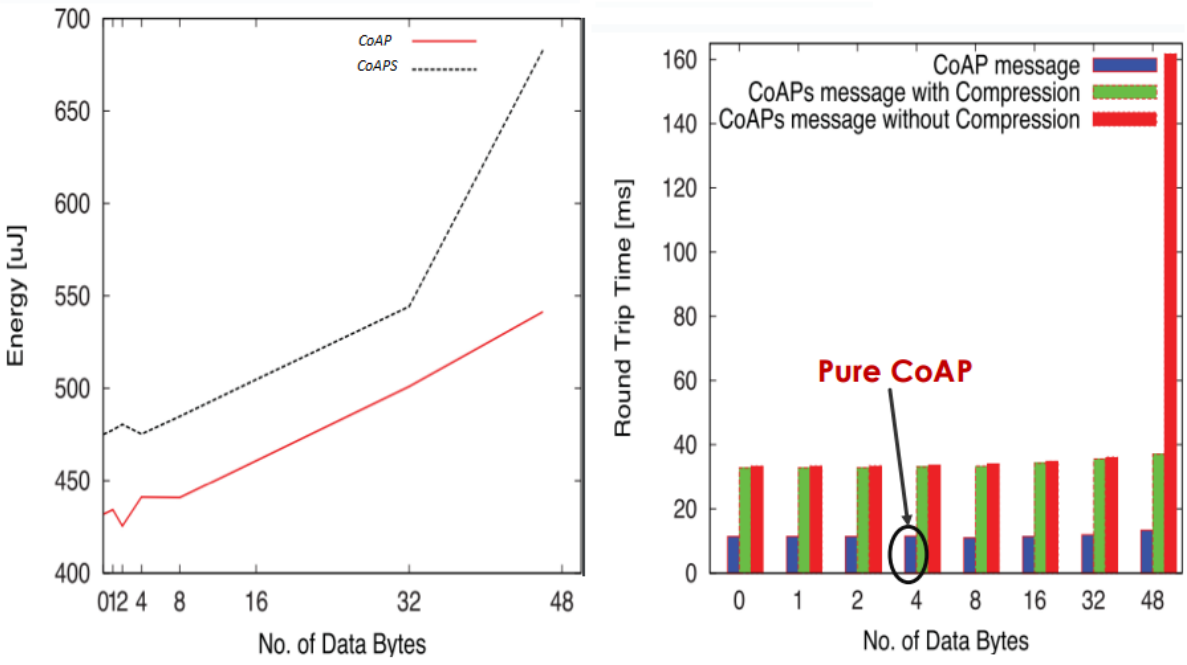

Ilustración 26. Comparación CoAP y CoAPs. [19]

Como era de esperar, se puede observar que tanto la cantidad de energía consumida, como el valor RTT han aumentado con el uso de DTLS frente al uso de CoAP puro.

En la primera gráfica se puede ver como a partir de los 32 Bytes de información enviada, se produce una fuerte subida de energía por lo que desde esa cantidad podría no ser rentable emplear CoAPs.

A su vez, en la segunda gráfica se puede observar que el tiempo de transmisión del mensaje usando CoAPs se dobra respecto a utilizar simplemente CoAP.

En un entorno real habría que analizar si saldría rentable añadirle a CoAP seguridad o de lo contrario no porque tanto nuestros dispositivos como el sistema no podría rendir de forma correcta al añadírsela.

Otros datos interesante a estudiar para el uso de DTLS con CoAP es el tiempo que tardaría cada uno de los diferentes niveles de seguridad que tiene CoAP. Para esta prueba se muestra una gráfica con seis ejemplos donde cada uno de ellos es un caso distinto de seguridad.

Estos ejemplos de seguridad son:

- **NoSec**: DTLS está deshabilitado.
- **PreSharedKey**: DTLS está habilitado, se utiliza como *ciphersuit* TLS_PSK_WITH_AES_128_CCM_8, sin autenticación del cliente, y sin fragmentación de los mensajes de *handshake*.
- **Certificate I**: DTLS está habilitado, se utiliza como *ciphersuit* TLS_PSK_WITH_AES_128_CCM_8, sin autenticación del cliente, y sin fragmentación de los mensajes de *handshake*.
- **Certificate II**: DTLS está habilitado, se utiliza como *ciphersuit* TLS_PSK_WITH_AES_128_CCM_8, con autenticación del cliente, y sin fragmentación de los mensajes de *handshake*.
- **Certificate III**: DTLS está habilitado, se utiliza como *ciphersuit* TLS_PSK_WITH_AES_128_CCM_8, con autenticación del cliente, y con fragmentación de los mensajes de *handshake*.
- **CoAPs**: DTLS está habilitado, pero ya se realizó anteriormente a esta petición un *handshake* por lo que la sesión está activa.

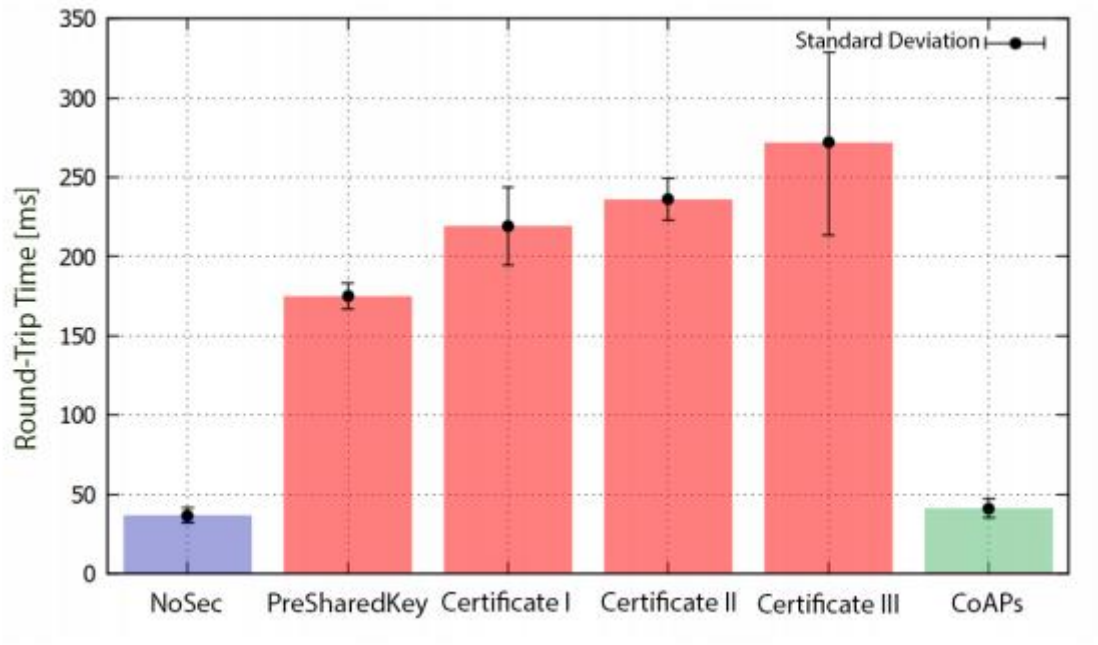

Ilustración 27. Comparación niveles de seguridad. [19]

En esta gráfica podemos ver que en la relación entre el RTT y los niveles de seguridad intermedios no hay mucha diferencia en cuanto a tiempo. Cierto es que a la hora de mandar grandes cantidades de datos esta diferencia sí que se podría notar, pero al no usarse CoAP, generalmente, para este tipo de utilidades, podría ser conveniente utilizar el nivel más seguro, en caso de requerir seguridad.

10) Multicast Group Communication

Como ya se ha comentado anteriormente, CoAP puede realizar peticiones multicast a través de IP. Esto permite que una única petición sea transmitida a un gran número de receptores, y de forma inversa, un cliente CoAP puede recibir a la vez representaciones de fuentes de un gran número de servidores utilizando CoAP sobre IP multicast.

Un ejemplo de esto podrían ser unas bombillas en una habitación con un grupo de comunicación CoAP la cuales podrían encenderse o apagarse a la vez solo utilizando IP multicast.

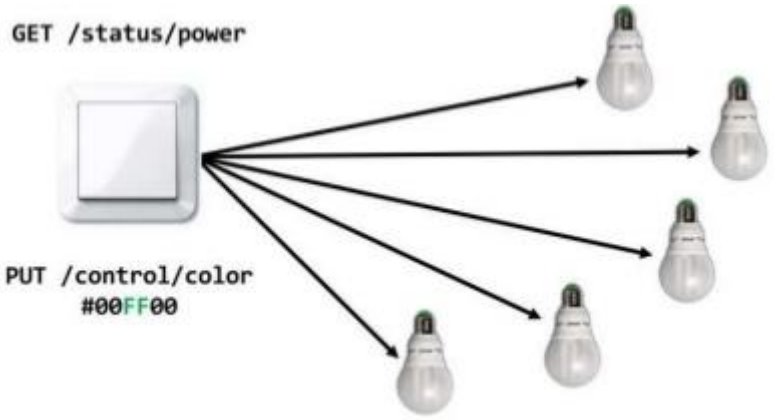

Ilustración 28. Ejemplo caso multicast CoAP.

En cuanto a la seguridad, hay que tener en cuenta que no es posible usar multicast con DTLS aunque existen otras formas de añadir seguridad.

11) Service and Resource Discovery CoAP

Para llevar a cabo un descubrimiento de servicio (**Service discovery**), el cliente debe saber el *endpoint* usado por el servidor.

Un servidor es descubierto por el cliente cuando este último aprende la URI que referencia una fuente en el espacio de nombres del servidor. Alternativamente los clientes puedes utilizar la opción multicast de CoAP y con las direcciones de todos los nodos CoAP encontrar servidores.

El **Resource discovery** permite a los recursos hospedados en los servidores de CoAP ser descubiertos enviando peticiones GET a ***/.well-known/core***. Esto revelará una lista de todos los recursos conocidos dentro del directorio de recursos.

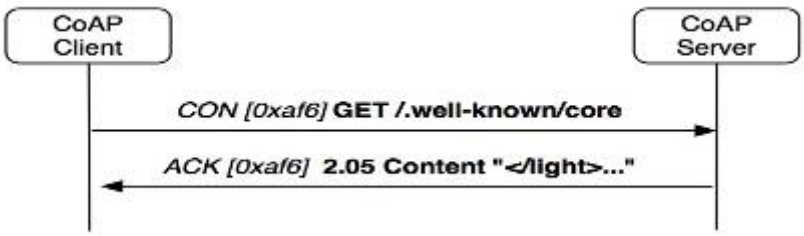

Ilustración 29. Ejemplo de Resource discovery. [13]

El descubrimiento de recursos usado por un punto final de CoAP es realmente importante para las aplicaciones *machine-to-machine* donde no se emplean personas para controlar el proceso.

12) Observable

El protocolo CoAP en cuanto a su característica observable se basa en la tecnología **Observer pattern**. En este diseño, existen los componentes "observers" los cuales serían clientes que se registran en un recurso en concreto con el objetivo de recibir una versión actualizada de un valor contenido en este recurso.
El otro componente es el "subject". En el contexto de CoAP, este elemento es un recurso alojado en un servidor el cual puede ir variando continuamente.

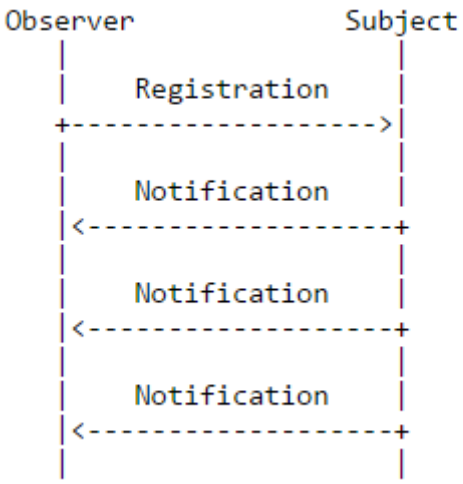

Ilustración 30. Ejemplo observable. [4]

Un cliente llega a ser un Observer cuando se incluye la opción de observable (Observer option) en una petición. Desde ese momento el cliente se subscribirá a un recurso observable haciendo una petición GET (alojado en un servidor, por ejemplo), y empezará a recibir notificaciones hasta que se cumpla una de las siguientes condiciones: un mensaje confirmable no sea respondido (*unacknowledgement*), el cliente envíe un mensaje explícito de que ya no quiere recibir más notificaciones, o que el valor *Max-Age* de la última notificación se exceda.

13) MQTT

No solo podemos pensar que existe un único protocolo preparado para este tipo de entornos.

A la vez que CoAP, se han creado más protocolos que tratarán de solventar las necesidades que están surgiendo. Otro ejemplo de estos es **MQTT**.

MQTT (MQ **Telemetry Transport**) es un protocolo de mensajería de publicación/subscripción diseñado para comunicaciones *Machine-to-Machine* ligeras. Originalmente fue diseñado por IBM y ahora es *open source*. [16]

Este nuevo protocolo y CoAP comparten varias características, como son:

- Ambos son *Open Source*.

- Tienen mejores características que HTTP para el tipo de entornos del IoT.
- Tienen mecanismos para enviar mensajes asíncronos.
- Trabajan sobre IP.

A continuación se va a mostrar una tabla donde se puede ver una pequeña comparación entre algunas de sus características:

	CoAP	MQTT
Communications Model	Request-Response, or Publish-Subscribe	Publish-Subscribe
RESTful	Yes	No
Transport Layer Protocol	UDP (TCP can be used)	TCP (UDP can be used; MQTT-SN)
Header	4 bytes	2 bytes
Number of Message Types	4	16
Messaging	Asynchronous & Synchronous	Asynchronous
Scalability	Complex	Simple
Security	DTLS	SSL/TLS
QoS options	Yes (Confirmable/Non confirmable messages)	Yes (3 levels)
Encoding	Binary	Binary
Dynamic discovery	Yes	No

Ilustración 31. Comparación CoAP y MQTT. [17]

Una vez ya presentadas las características de cada uno, llega la hora de juzgarlos y ver realmente qué protocolo es mejor.

Realmente ninguno de los dos podría decirse que supera al otro en cuanto a prestaciones. Cada uno de ellos se ajustará de una mejor manera a un tipo de entorno.

Pongamos como ejemplo un caso en el que se asume que se está utilizando un dispositivo de bajas prestaciones y se quiere elegir el mejor. En general si lo único que queremos es recuperar información de una red con recursos limitados, CoAP podría ser una mejor opción. Sin embargo, si una red de dispositivos (sensores, controladores,…) necesitaran estar al tanto de todo lo que están haciendo los demás, enviar mensajes a todos podría ser demasiado costoso. Para un dispositivo es más barato enviar un mensaje a un servidor, el cual no tenga recursos limitados, y así ahorrar energía. En este caso saldría más rentable utilizar MQTT.

Por esto que se ha explicado, no se puede decir que protocolo es mejor en general, solamente ver cuál es el que mejor se adapta a lo que queremos diseñar.

Una vez ya vistas las propiedades de CoAP y lo que se puede realizar con ellas, hay que hablar sobre los softwares ya creados sobre este protocolo y el elegido para desarrollar este TFG.

Existen varios software de desarrollo para este protocolo. Se pueden encontrar software hechos en muchos lenguajes de programación como Java, C, Phython, Android, etc, los cuales algunos están especializados en diferentes tareas del diseño.

JAVA	C / C#	Otros lenguajes
Californium	libcoap	Californium (Android)
nCoAP	CoAP.NET	Node-coap (JavaScript)
Leshan	-	txThings (Phython)

Tabla 3. Ejemplo de software de desarrollo.

Para este TFG, se va a utilizar el software **Californium para Java.** Se ha elegido por las comodidades que ofrece Java al ser un lenguaje de alto nivel, por ser **Open Source** y por la cantidad de opciones de diseño que da Californium ofreciendo varios softwares especializados en distintas funcionalidades de CoAP como son el multicast, DTLS para CoAP,…

2.4 CALIFORNIUM

Californium (Cf) es una implementación **open source** de CoAP perteneciente a Eclipse. Está implementado en Java y contiene un API RESTful con todas las características del protocolo CoAP.

Californium proporciona al usuario un **framework intuitivo** y fácil de usar para interactuar con los endpoints de CoAP o proporcionar servicios específicos. De esta forma el usuario no tendrá que encargarse de lidiar con las partes internas como las retransmisiones de mensajes y las transferencias en bloque.

Desde el punto de vista del cliente, una operación de RESTfull puede ser realizada únicamente con la creación de una petición GET, POST, PUT o DELETE, especificando el URI del punto final de destino, seguido de la carga útil y otras opciones en caso de ser necesario. Tras la petición, las correspondientes respuestas pueden ser procesadas de forma sincrónica o asincrónica.

La configuración de un servidor CoAP utilizando Californium, no requiere nada más que definir los recursos (resources) que lo van a formar proporcionando subclases que implementen los maneadores de peticiones GET, POST, PUT y/o DELETE. Es decir, no es necesario crear de cero ningún mensaje.

Sin embargo, las propiedades del protocolo podrían permitir que el usuario personalizara un mensaje adaptándose así a las sus necesidades.

Californium ha sido recientemente re-implementado y ahora está principalmente enfocado a la creación de servicios para aplicaciones del Internet of Things.

El proyecto se divide en 5 subproyectos:

- **Californium (Cf):** es el framework central con la implementación del protocolo para construir aplicaciones sobre el IoT.
- **Scandium (Sc):** provee seguridad, implementa DTLS.
- **Actinium (Ac):** es el app-server de Californium y se emplea para realizar mashups en JavaScript.
- **CoAP tolos**
- **Connector:** abstrae a los diferentes transportes.

Todo este proyecto se encuentra íntegramente en **GitHub**.

Ahora se va a hablar brevemente sobre el entorno de desarrollo Java y posteriormente se verán las librerías que se han utilizados para la realización del proyecto.

Para poder desarrollar una aplicación en Java se necesita un entorno de desarrollo con Eclipse. Además se deberá descargar el **kit de desarrollo JDK (Java Development Kit).**

Eclipse es una plataforma de desarrollo de código abierto basada en Java. Por si misma, es simplemente un marco de trabajo y un conjunto de servicios para la construcción del entorno de desarrollo de los componentes de entrada. Afortunadamente, Eclipse tiene un conjunto de complementos, incluidas las Herramientas de Desarrollo de **Java (JDT)**. [14]

JDK es un conjunto de herramientas (programas y librerías) que permiten desarrollar (compilar, ejecutar, generar documentación, etc.) programas en lenguaje Java.

3 DISEÑO Y DESARROLLO

3.1 Planteamiento y diseño

Una vez ya explicadas las tecnologías que se han empleado en este proyecto se va a proceder a explicar de forma detallada cómo se ha llevado a cabo el diseño y desarrollo de este.

Inicialmente se va a comenzar explicando el entorno que se quiere simular. A continuación se va a mostrar un ejemplo que representará de forma general un sistema de Internet of Things utilizando el protocolo CoAP. Este entorno podrá ser modificado según sea necesario, cambiando las propiedades de cada uno de los bloques o incluso añadiendo nuevos en un futuro.

En la siguiente imagen se pueden observar dos tipos de actores en el sistema, los clientes y los servidores:

- **Clientes**. Simulan ser objetos inteligentes del Internet de las Cosas. Como por ejemplo un sensor de temperatura que envía datos constantemente o una bombilla.

- **Servidor CoAP**. Este elemento se encarga de gestionar las peticiones realizadas por los clientes. Una vez las reciben, estas son procesadas y, en caso de requerir una contestación, el servidor enviaría una respuesta al cliente. Este caso se daría por ejemplo cuando la petición es CON, es decir, requiere un ACK como respuesta.

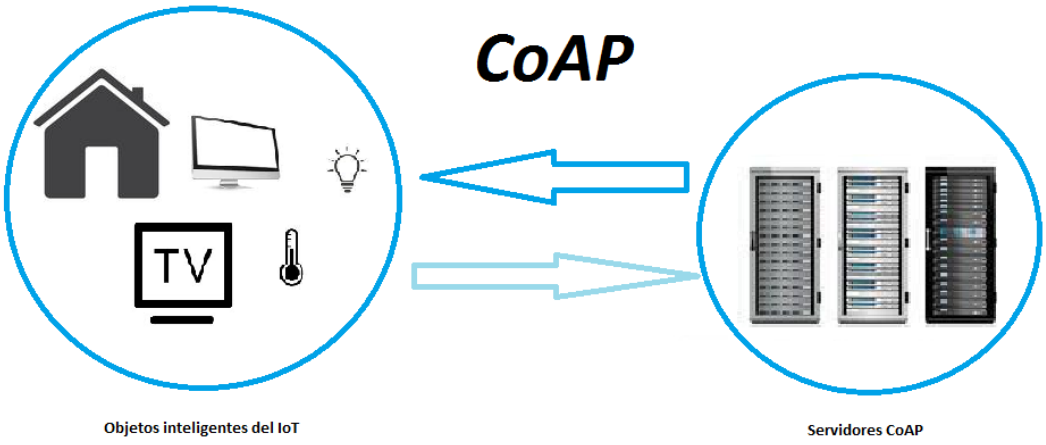

Ilustración 32. Ejemplo de un entorno con CoAP.

Como se ha explicado anteriormente y como se puede ver en la imagen, los objetos y el servidor interactúan entre sí enviándose información mediante CoAP. Los diferentes clientes realizan peticiones al servidor el cual las procesa y las contesta.

A continuación, se va a explicar que información se intercambia en cada paso de la comunicación:

- **Paso 1: Petición del cliente al servidor.**

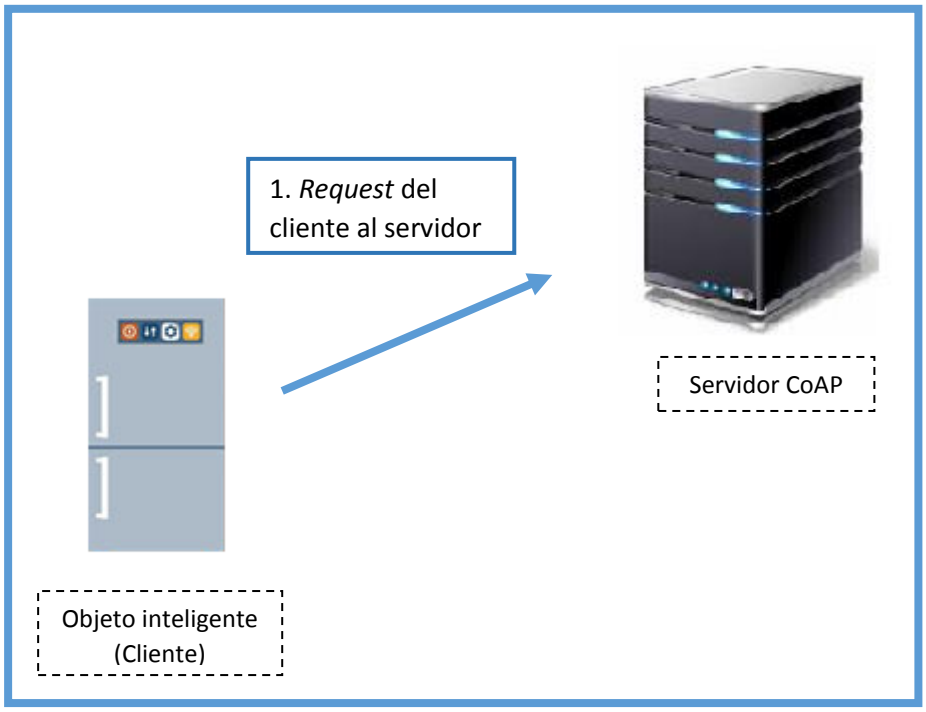

Ilustración 33. Petición cliente al servidor.

Una vez esté el servidor arrancado los diferentes clientes le harán peticiones siguiendo una estructura parecía a la empleada en el protocolo HTTP.
Cada uno de los objetos deberá tener programada las peticiones que podrá realizar y poder manejar las contestaciones realizadas por el servidor.

- **Paso 2: Respuesta del servidor al cliente.**

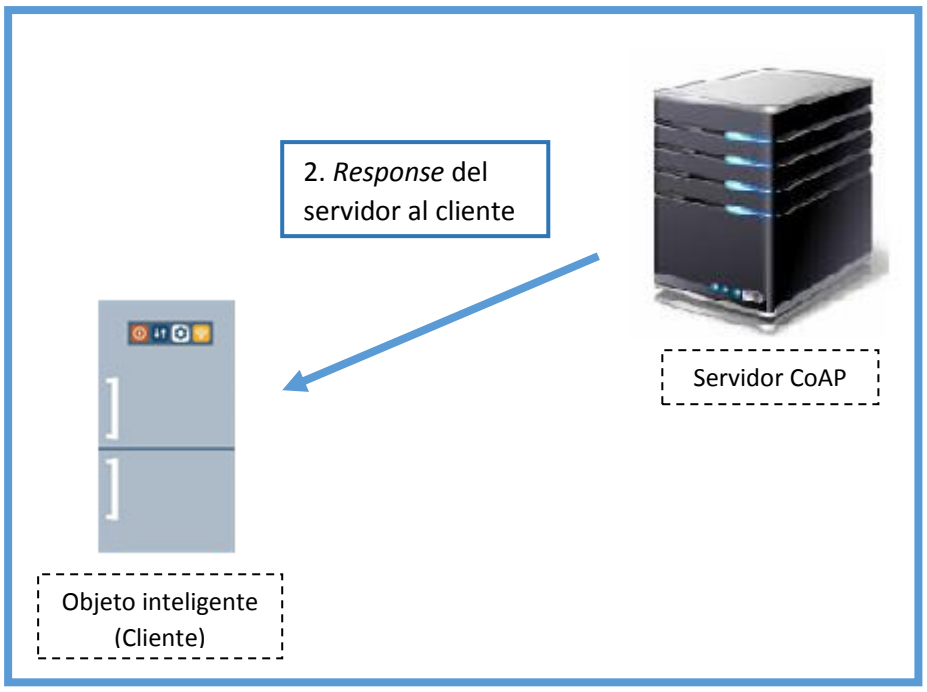

Ilustración 34. Respuesta del servidor al cliente.

El servidor deberá de estar preparado para entender, procesar y, si es necesario, responder a las posibles peticiones que lleguen por parte de los clientes. Tendrá que machear la petición del cliente con los posibles recursos que tendrá incorporadas en su software y a los que le podrá que dar servicio. Una vez encontrado el manejador encargado de esa petición, y en caso de que se requiera una respuesta (no siempre se necesitará), se enviará al cliente, o un ACK indicando que todo ha ido bien, o la respuesta con el contenido que se requería.

- **Paso 3: Respuesta del cliente.**

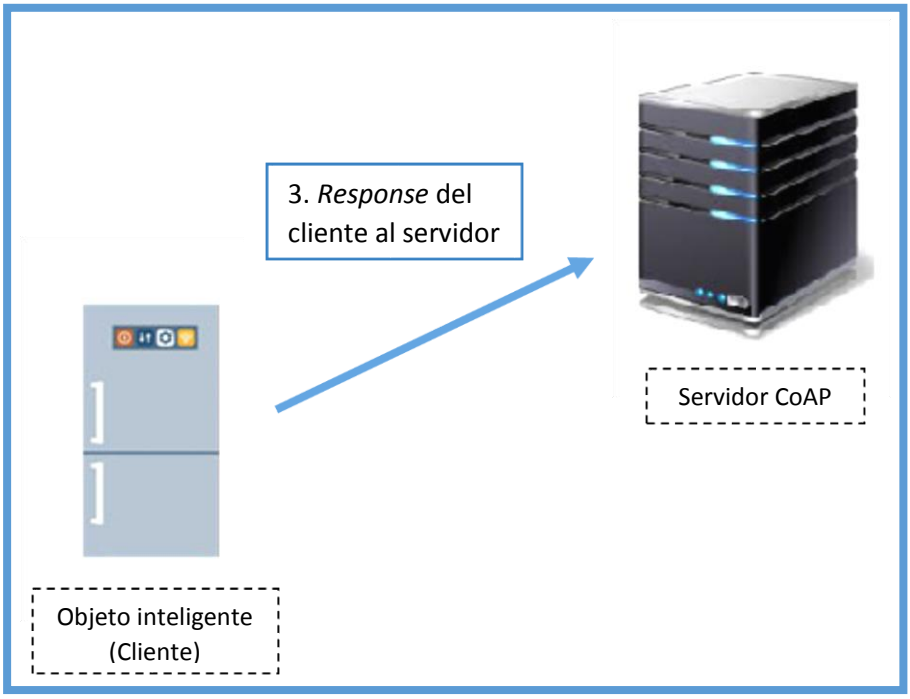

Ilustración 35. Respuesta del cliente.

En caso de ser necesario el cliente responderá al servidor ya sea o con un ACK confirmando que ha recibido la respuesta o con cualquier otra contestación que tenga programada para dicha respuesta del servidor.

La imagen que se muestra debajo simula ser un ejemplo de intercambio de mensajes. En ella el cliente envía una petición GET indicada como que debe ser confirmada (CON). Posteriormente el cliente contesta con un ACK.

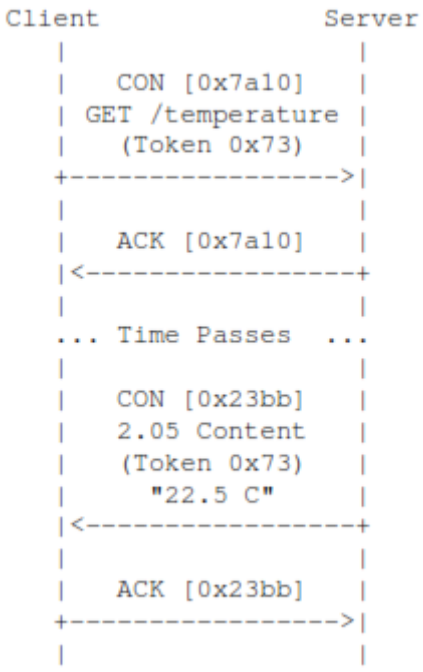

Ilustración 36. Ejemplo intercambio de mensajes. [4]

3.2 Desarrollo de la solución

3.2.1 Requisitos previos al desarrollo

Después de dejar claro el problema planteado y el diseño, se va a proceder a explicar el desarrollo de la solución y las herramientas que proporcionan tanto las librerías utilizadas como el entorno de desarrollo.

Para comenzar se necesita contar con un equipo que posea un sistema operativo que cuente con el JDK de Java instalado.

En este trabajo se ha empleado el sistema operativo Windows 8.1 en el que se ha instalado la **versión 1.8.0_77 de Java**. Como entorno de desarrollo se ha utilizado la **versión "Mars.2 Release (4.5.2)" de Eclipse** (Eclipse Java EE IDE for Web Developers).

En cuanto a las librerías que se requerían para desarrollar el proyecto se ha decidido que sean importadas utilizando la herramienta **Maven**.

Maven es una herramienta software usada para la construcción y el manejo de un proyecto Java cuyo modelo de configuración está basado en el formato XML. Su principal objetivo es hacer que los desarrolladores dediquen el menor tiempo posible en el proceso de creación y construcción de un proyecto.

Para alcanzar este objetivo hay varias áreas que Maven intenta abordar: [15]

- Hacer el proceso de construcción fácil.
- Proporcionar un sistema de construcción uniforme.
- Proporcionar información de proyecto de calidad.

- Proporcionar directrices para mejores prácticas de desarrollo.
- Permitir la migración del proyecto.

Maven es un proyecto de nivel superior de Apache Software Foundation. Utiliza POM (Project Object Model) para describir el proyecto de software a construir, sus dependencias de otros módulos y componentes externos, y el orden de construcción de los elementos. Posee unos objetivos predefinidos con los que desarrollar ciertas tareas claramente marcadas, como la compilación del código y su empaquetado.

Maven está diseñada para ser usada en la red. Descarga de forma dinámica plugins de un repositorio (el mismo repositorio que provee acceso a muchas versiones de diferentes proyectos Open Source en Java, de Apache y otras organizaciones y desarrolladores).

La librería principal que se ha necesitado para el desarrollo de este trabajo es la **versión 1.1.0 de Californium (Cf) core**.

Esta librería contiene diferentes paquetes de clases que formarán la estructura principal de la parte software del proyecto. En ella se incluyen las clases que harán posible las peticiones (Request) y las respuestas (Response), así como las clases que formarán las bases de los clientes y servidores.

En el caso de utilizar CoAP de forma segura utilizando DTLS se necesitará importar también los proyectos Java facilitados por Californium, “scandium” y “demo-certs”, en nuestro caso se ha utilizado la versión 1.1.0 de Californium:

- ***scandium***: este proyecto Java contiene las clases necesarias para implantar una capa de seguridad a CoAP utilizando DTLS. Permite utilizar claves secretas, claves públicas y certificados.
- ***demo-certs***: es el proyecto Java encargado de almacenar y proporcionar un ejemplo de claves que serán utilizadas posteriormente en los ejemplos diseñados.

Los enlaces de descarga de las librerías y proyectos mencionados arriba son los siguientes:

- Java:
  https://www.java.com/es/download/

- Eclipse:
  https://eclipse.org/downloads/

- Californium (Californium Core, Scandium, demo-cert):
  https://eclipse.org/californium/

3.2.2 Estructura de clases del proyecto

En cuanto a la estructura de clases del proyecto creado en Eclipse, se ha ido separando en paquetes, de tal forma que en cada paquete se haga la simulación de un entorno probando una característica específica de CoAP Californium. Estos paquetes son los siguientes:

- **californium.coap.example**: este paquete contiene el código fuente que simulará un entorno formado por un cliente (bombilla, sensor,…) y un servidor los cuales intercambiarán mensajes utilizando el protocolo CoAP.

- **californium.scandium.example**: en este paquete se simulará un entorno similar al anterior pero se utilizará el paquete de Californium llamado Scandium para dotar de seguridad mediante DTLS al intercambio de mensajes entre cliente y servidor.
- **californium.observable.example**: este último paquete se encarga de representar la simulación de un entorno en el que se utilizarán recursos observables, es decir, un cliente CoAP observa un recurso en un servidor.

Ya definidos todos los paquetes que se encuentran en el proyecto, hay que tener en cuenta que cada uno de estos posee un método *main* encargado de la ejecución del entorno que se ha querido crear para cada paquete.

A continuación se muestra en la siguiente imagen la estructura del proyecto en Eclipse:

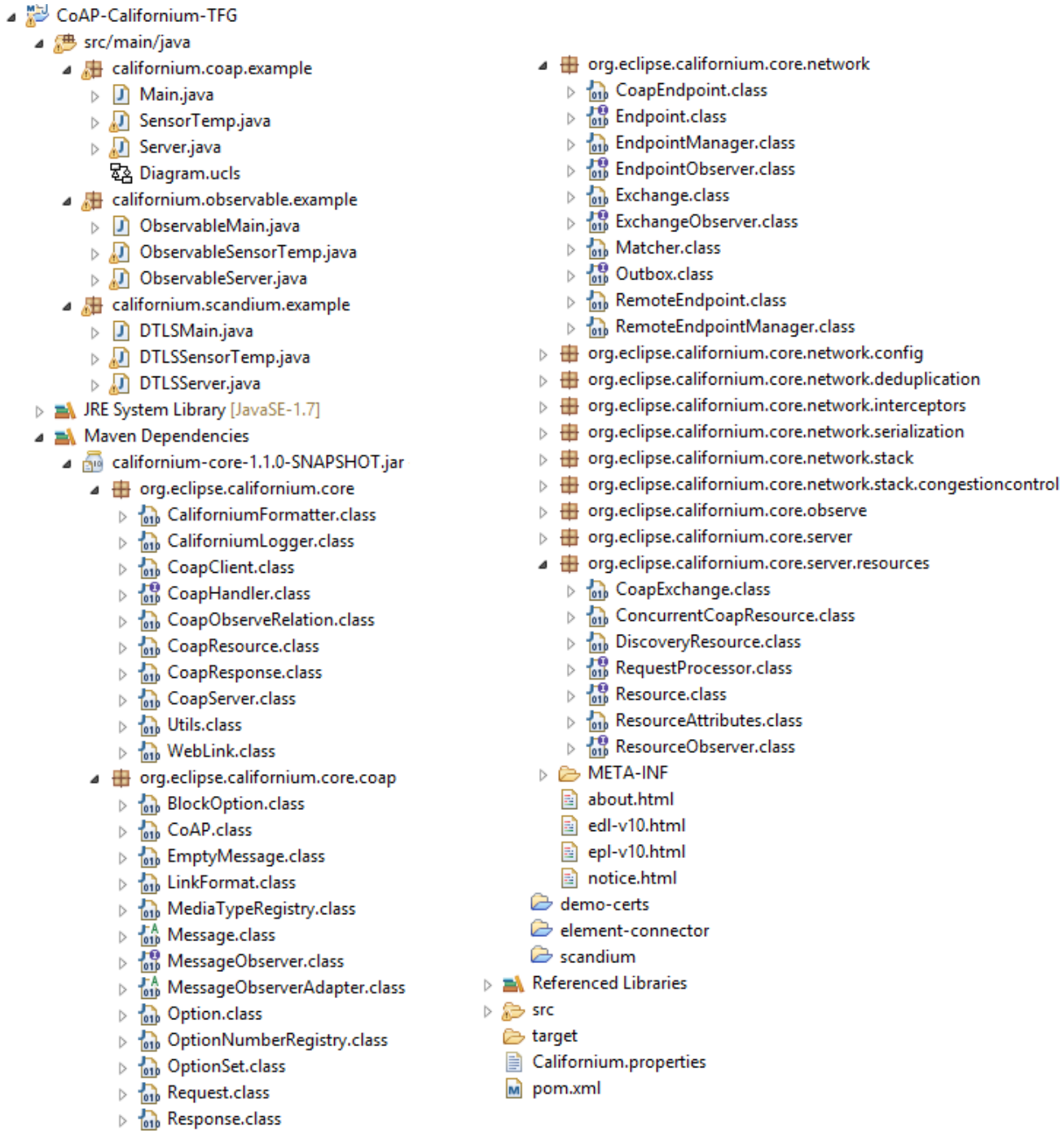

Ilustración 37. Estructura del proyecto en Eclipse.

En la imagen se puede ver el nombre del proyecto "CoAP-Californium-TFG", los diferentes paquetes que lo componen y las clases que pertenecen al *jar* de *californium-core*. También se observa que están importados tres proyectos, "demo-certs", "element-connector" y "scandium", todos estos pertenecientes al paquete software de Californium.

- **Bloque entorno CoAP**

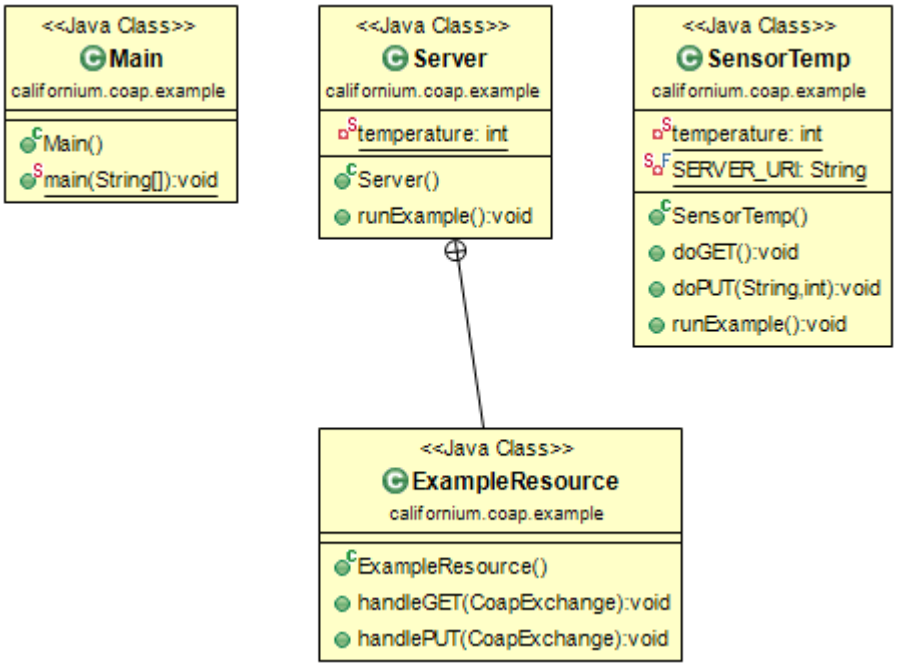

Ilustración 38. Bloque CoAP.

Como ya se ha comentado anteriormente, este bloque es el encargado de simular un entorno de transmisión de datos utilizando el protocolo CoAP.

Está formado por un cliente y un servidor los cuales intercambiarán mensajes tratando se emular un sistema real. El cliente realizará peticiones al servidor CoAP el cual responderá con los datos solicitados.

Para la ejecución del paquete se han creado 3 clases Java que lo definen:

- **Server.java**: Es la clase encargada de simular un servidor CoAP. Se ha necesitado crear en ella una clase (TempResource) encargada de añadir al servidor los diferentes manejadores que harán que pueda responder a las peticiones que vendrán por parte del servidor.

  Para este ejemplo se han diseñado dos manejadores (handles):

  - handleGET: se encargará de devolver el valor de la temperatura que se encuentra guardada en el servidor.
  - handlePUT: guardará en el servidor el valor de temperatura enviado por el cliente.

- **SensorTemp.java**: Esta clase simula ser un cliente y en concreto para este ejemplo un sensor de temperatura. Este sensor tendrá la opción de almacenar un valor *temperature* el cual puede ser modificado por sí mismo o por el servidor. Por ello este cliente realizará dos tipos de peticiones (Request):

- doGET: en este método el cliente pide al servidor el valor de temperatura que él tiene guardado.
- doPUT: el cliente pide al servidor que cambie su valor de temperatura por el que tiene el sensor.

- **Main.java:** Es el encargado de ejecutar el ejemplo de simulación del paquete. En el primero el cliente pedirá al servidor que le de su valor para temperature y posteriormente el sensor le enviará un nuevo valor.

- **Bloque entorno CoAP seguro**

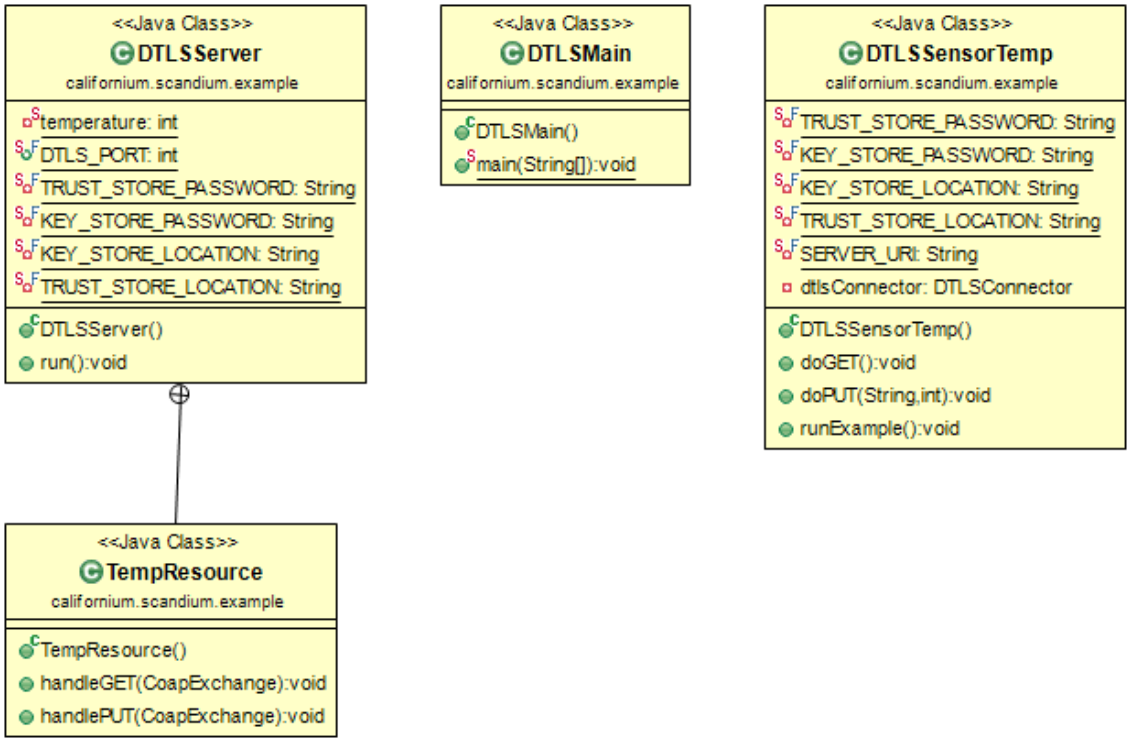

Ilustración 39. Entorno CoAP con DTLS.

Este entornó realizará las mismas funciones que el paquete anterior pero ahora además irá protegido por el protocolo de seguridad DTLS. De esta forma se conseguirá dotar al protocolo COAP de la capa de seguridad que por defecto no dispones. Se realizará el mismo ejemplo de transmisión de datos entre un cliente y un servidor CoAP.

Este paquete está definido por tres clases diferentes:

- **DTLSServer.java:** Simula ser un servidor CoAP el cual utiliza DTLS para mejorar la seguridad. A este servidor se le proporcionará una clave secreta, un certificado y un par de claves formado por una privada y una pública. De esta forma podrá elegir entre estas formas de seguridad para realizar la comunicación con el cliente. Para el cifrado utilizará AES 128.

Por otra parte al igual que en el paquete anterior el servidor utilizará las Resources para crear las posibles respuestas a las peticiones realizadas por el cliente. Como posibles manejadores de respuestas se utilizarán las mismas que se han empleado en el servidor CoAP anterior: handleGET y handlePUT.

- **DTLSSensorTemp.java:** Esta clase será la encargada de simular ser un cliente CoAP y en este caso, al igual que en el anterior, se trata de un sensor de temperatura capaz de modificar el valor temperature del servidor y de permitir que el servidor modifique su valor temperature.

  De la misma forma que el servidor, utiliza DTLS para la transmisión de datos utilizando el mismo tipo de claves que este y también usará AES128 para el cifrado.

  Este cliente realizará las dos mismas peticiones ya explicadas para el paquete anterior: doGET y doPUT.

- **DTLSMain.java**: Esta clase es la encargada de ejecutar el paquete y así realizar la simulación de una transmisión de datos con CoAP y DTLS. Para ello no se necesitará nada más que llamar a los dos métodos de arranque correspondientes. El ejemplo de simulación será el mismo que el realizado para el paquete anterior.

- **Bloque observable**

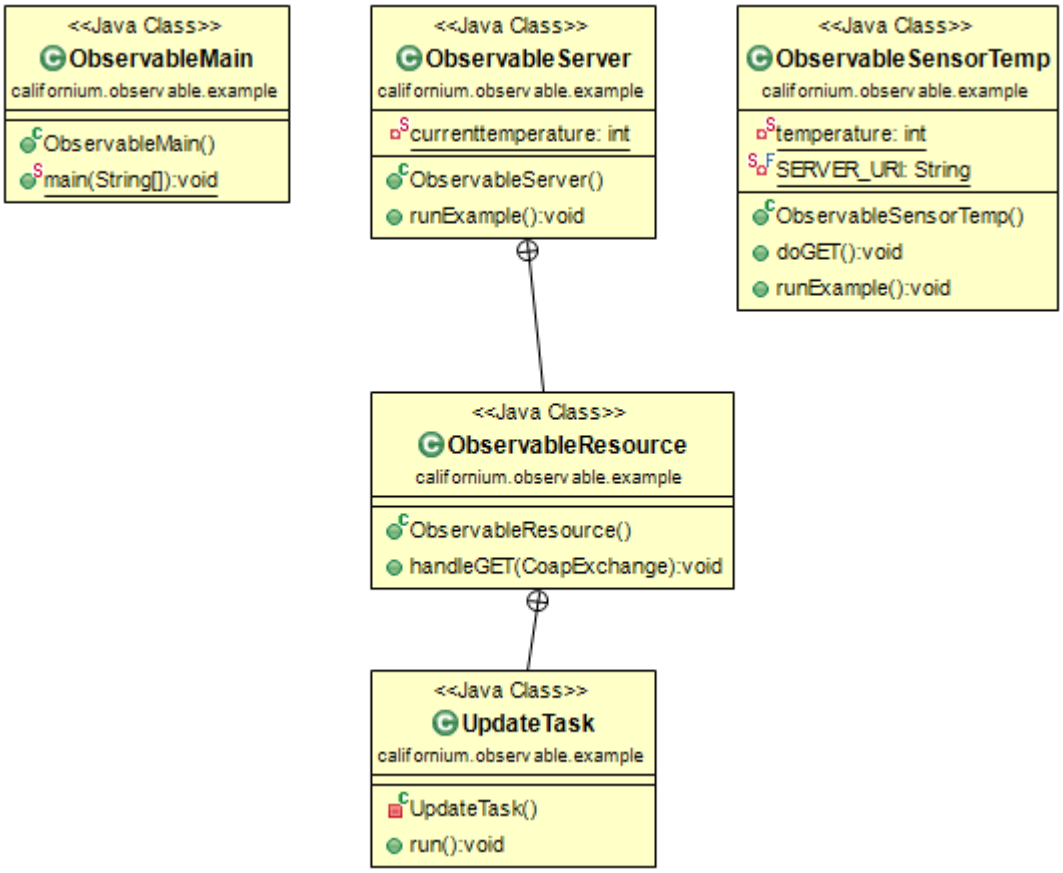

Ilustración 40. Entorno CoAP con Observable Resources.

La simulación llevada a cabo en este paquete se aleja ligeramente del esquema utilizado en los anteriores donde el cliente hacía una petición y el servidor le daba una única contestación. En este caso el cliente realizará una Request observable indicándole al servidor que está interesado en tener una representación actual de una fuente a lo largo de un tiempo determinado.

Este paquete lo forman 3 clases:

- **ObservableServer.java**: Esta clase simula ser un servidor CoAP el cual utiliza la tecnología Observable. De esta forma este espera a recibir una petición (indicada anteriormente como petición observable) por parte del cliente. Una vez reciba la petición de un recurso observable, procederá a responder al cliente con el valor que este le ha requerido. Tras ser enviada esta primera respuesta y de forma periódica el servidor irá enviado respuestas al cliente con el valor actual en ese instante de tiempo. Este periodo se podrá modificar de acuerdo a lo que interese en cada ocasión.

  Las respuestas del servidor podrán ser configuradas como CON-Responses o como NON-Responses.

Excluyendo la parte del código encargada de dar el carácter Observable al servidor, la Resource empleada en este actuará de forma similar a como lo hace la explicada en los paquetes anteriores. Contará con un manejador encargado de controlar las peticiones GET hechas por el cliente (handleGET) y un manejador encargado de las peticiones PUT que actualizarán el valor del servidor (handlePUT).

- **ObservableSensorTemp.java**: Esta es la clase que simula ser un cliente CoAP el cual realizará una petición Observable, es decir, el cliente hará de Observer. Para esta petición se deberá añadir a la trama la opción Observe Option.

  La parte de código que no esté relacionada con dar el carácter observable a la petición será semejante al explicado en los paquetes anteriores para los clientes. Este cliente dispondrá de dos tipos de peticiones, un GET (doGET) y un PUT (doPUT).

- **ObservableMain.java**: Esta clase será la encargada de ejecutar el paquete y de esta forma realizar la simulación del entorno empleando la tecnología Observable. Para ello no se necesitará nada más que llamar a los dos métodos de arranque correspondientes. El ejemplo de simulación será el mismo que el realizado para los paquetes anteriores pero empleando observables.

4 ESTUDIO Y ANÁLISIS

Una vez ya explicadas las herramientas y tecnologías utilizadas para el diseño y desarrollo de los diferentes entornos CoAP, utilizando como soporte las librerías que proporciona Californium, se realizará el estudio y análisis de las distintas características de éste que han sido probadas.

Para comenzar se analizarán los resultados obtenidos en las simulaciones de los entornos anteriormente planteados visualizando el intercambio de mensajes que se ha ido produciendo en éstas, y posteriormente se estudiarán y analizarán diferentes propiedades de las tecnologías empleadas.

COAP

El primer paquete del proyecto que se creó y ejecutó fue el encargado de crear un entorno simple de CoAP utilizando nada más que las propiedades más básicas que la herramienta Californium proporciona. Este es el paquete **californium.coap.example.**

Tras ejecutar la clase Main.java de este paquete conseguimos poner en marcha la simulación. Hay que recordar que esta simulación consistía en crear un entorno formado por un cliente, en este caso un sensor de temperatura, y un servidor, los cuales intercambiaban mensajes.

En la siguiente imagen se pueden ver los primeros datos que se obtienen en consola al simular este primer paquete ejecutando la clase *main*:

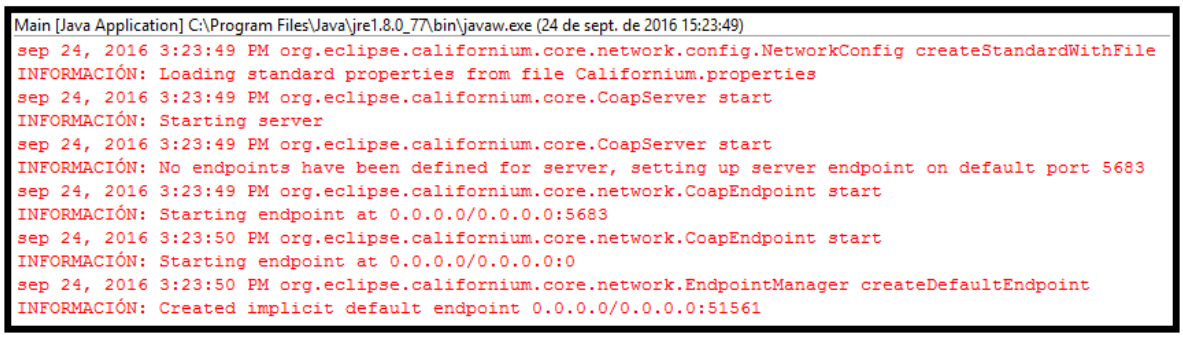

Ilustración 41. CoAP - Captura.

En esta captura de pantalla podemos ver como lo primero que se hace es arrancar el servidor CoAP y el cliente, poniéndose este último a escuchar en la dirección IP 0.0.0.0 y el puerto 5683, el cual es el empleado por defecto por CoAP. Que se utilice esa dirección IP se debe a que cuando un servicio escucha por conexiones en la dirección 0.0.0.0 significa que el servicio escuchará conexiones en todas las direcciones IP disponibles en la computadora.

Una vez arrancado el servidor, ya estará operativo para recibir cualquier petición por parte de clientes o *endpoints*.

A continuación podemos ver una imagen donde se muestran los resultados obtenidos por parte del cliente al realizar éste una petición GET al servidor pidiéndole la temperatura:

```
CLIENT: Temperature Sensor
---[Coap Request GET done]---

SERVER
==[ CoAP Response ]=============================================
MID     : 40458
Token   : 624bca6ec26d
Type    : ACK
Status  : 2.05
Options: {"Content-Format":"text/plain"}
Payload: 2 Bytes
----------------------------------------------------------------
24
================================================================
```

Ilustración 42. CoAP - Petición GET.

Se puede ver como el sensor realiza la petición y obtiene como respuesta una trama bien definida donde se puede observar perfectamente los diferentes valores de la cabecera (MID, Token, Type,…) y el cuerpo, en este caso un payload de 2 Bytes donde se indica el valor de la temperatura, 24 grados centígrados.

En la siguiente imagen se muestra un caso donde el cliente realiza una petición PUT indicándole al servidor que modifique su valor de temperatura por el que lleva la petición en el campo payload:

```
CLIENT: Temperature Sensor
---Request PUT done with payload = 12---


SERVER
==[ CoAP Response ]=============================================
MID     : 40459
Token   : f09544973f
Type    : ACK
Status  : 2.04
Options: {"Content-Format":"text/plain"}
Payload: 2 Bytes
----------------------------------------------------------------
12
================================================================
```

Ilustración 43. CoAP - Petición PUT.

En la respuesta se pude apreciar que el contenido del campo *Status* es “2.04”, que como ya se explicó, quiere decir que el valor ha sido cambiado.

Una vez ya explicado el qué es CoAP, cómo funciona, y vistos los resultados obtenidos al realizar la simulación de este protocolo, vamos a proceder a su análisis.

CoAP fue creado con el fin de satisfacer las nuevas necesidades que iban surgiendo debido al incremento de dispositivos conectados entre sí (Internet of Things). A diferencia de otros protocolos ya existentes encargados de la transmisión de datos, CoAP se especializó en dispositivos de bajas prestaciones. Es decir, debido a características como una cabecera muy pequeña, CoAP daba la posibilidad a un *endpoint* de realizar una comunicación con otro sin apenas requerir energía ni memoria.

Un ejemplo de protocolo a nivel de aplicación, al cual CoAP puede quitar el puesto en cuanto a la transmisión de mensajes es HTTP.

Estos dos protocolos **comparten el modelo de petición/respuesta** utilizando ambos el mismo diseño RESTful con las opciones GET/PUT/POST/DELETE. El problema de HTTP para este tipo de entornos, es el gran tamaño que tiene su cabecera, lo que hace que por cada mensaje enviado el emisor gaste demasiada energía. Esta cabecera en HTTP al ser texto estructurado, es más fácil de leer para el ser humano, pero no para la máquina, situación que para al contrario en CoAP.

Otra ventaja que hace a CoAP más eficiente para un entorno del Internet of Things, es la utilización de UDP. UDP al no necesitar que exista un *handshake* entre dos puntos para comenzar a intercambiar datos, hace que el proceso sea más rápido. Esta velocidad por decirlo de alguna forma no es gratuita. UDP no es un protocolo fiable y puede hacer que los paquetes enviados no lleguen a su destino, o lleguen en distinto orden. A pesar de este inconveniente que presenta UDP, CoAP ha trabajado para tratar de solucionarlo y ha introducido campos en su cabecera encargados de evitar ese tipo de situaciones como su detector de duplicado *Menssage ID* en la cabecera, con las peticiones confirmables o el uso de *tokens* para machear respuestas y peticiones.

Por todo esto CoAP es un protocolo perfectamente preparado para situaciones del Internet de las Cosas donde existe un gran número de dispositivos con bajas prestaciones. Aunque esto no quiere decir que CoAP vaya a sustituir a HTTP.

DTLS

A continuación se van a analizar y explicar los resultados obtenidos de la simulación del segundo paquete, **californium.scandium.example.**

Se recuerda que éste simulaba un entorno formado por un cliente (sensor de temperatura) y un servidor los cuales intercambiaban mensajes utilizando CoAP, y en concreto empleando DTLS, con el objetivo de aportar seguridad a estos.

A continuación vamos a ver una serie de imágenes las cuales no serán más que capturas de pantalla de la consola. Hay que destacar que la información mostrada en las imágenes únicamente es una parte de la que realmente se mostraba, ya que debido a su gran extensión, se ha decidido escoger los fragmentos más relevantes.

Como ya se ha visto anteriormente en CoAP, el primer paso a seguir en la simulación y el que

```
1 INFO [CoapServer]: Starting server -
(org.eclipse.californium.core.CoapServer.java:180) start() in thread main at
(2016-09-25 12:49:56)

 1 INFO [CoapEndpoint]: Starting endpoint at 0.0.0.0/0.0.0.0:5684 -
(org.eclipse.californium.core.network.CoapEndpoint.java:192) start() in
thread main at (2016-09-25 12:49:56)

13 CONFIG [DTLSConnector$Worker]: Starting worker thread [DTLS-Sender-
0.0.0.0/0.0.0.0:5684] -
(org.eclipse.californium.scandium.DTLSConnector$Worker.java:-1) run() in
thread DTLS-Sender-0.0.0.0/0.0.0.0:5684 at (2016-09-25 12:49:56)
```

Ilustración 44. CoAPs - Starting.

antes se muestra por pantalla es el arranque, tanto del servidor como del cliente, algo que se puede ver en la siguiente imagen:

Como se puede observar en la captura, ambos habilitan el puerto 5684 ya que éste es el utilizado por defecto para CoAP con DTLS.

Para la interconexión de mensajes entre dos puntos utilizando esta tecnología, es necesario que ambos realicen un *handshake* antes de comenzar con el intercambio de mensajes. El objetivo de esto es negociar la clave de seguridad, *cipher suites* y los métodos a usar.

Antes de analizar lo devuelto por pantalla para esta acción, se va a mostrar una imagen donde se podrá observar un resumen de cómo funciona este *handshake* entre un cliente y un servidor.

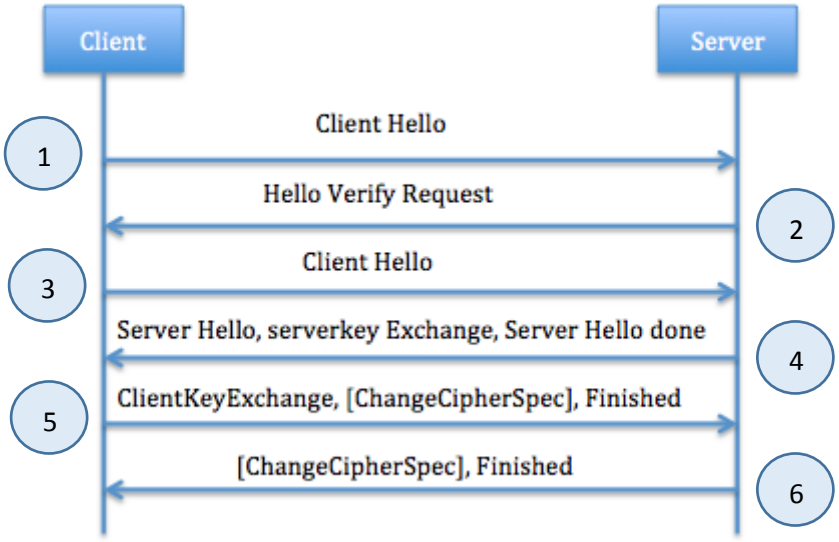

Ilustración 45. Resumen de un handshake en CoAPs. [18]

Una vez visto el resumen, se va a proceder a analizar paso por paso las acciones realizadas para llevarlo a cabo:

- **Paso 1** → el cliente envía una petición al servidor indicándole que quiere realizar una conexión con él. Esta acción podemos verla en consola en la siguiente captura:

```
18 FINE [DTLSConnector]: Received Handshake (22) record from peer
[/127.0.0.1:5684] -
(org.eclipse.californium.scandium.DTLSConnector.java:455)
processHandshakeRecord() in thread DTLS-Receiver-0.0.0.0/0.0.0.0:51643 at
(2016-09-25 12:49:56)
```

Ilustración 46. CoAPs – Cliente Hello.

- **Paso 2** → posteriormente podemos ver cómo el servidor responde al cliente indicándole que le vuelva a enviar la petición, pero esta vez con una cookie que este le envía. Esto lo hace el servidor para evitar ataques *DoS* (Denial of Service).

```
18 FINE [ClientHandshaker]: Processed HELLO_VERIFY_REQUEST (3) message with
sequence no [0] from peer [localhost/127.0.0.1:5684] -
(org.eclipse.californium.scandium.dtls.ClientHandshaker.java:372)
doProcessMessage() in thread DTLS-Receiver-0.0.0.0/0.0.0.0:51643 at (2016-
09-25 12:49:56)
```

Ilustración 47. CoAPs - Hello Verify Request.

- **Paso 3** → por lo comentado anteriormente, el cliente envía de nuevo la petición pero ahora con la cookie del servidor.

```
17 FINE [Connection]: Handshake with [localhost/127.0.0.1:5684] has been
started - (org.ec
lipse.californium.scandium.dtls.Connection.java:831) handshakeStarted() in
thread DTLS-Sen
der-0.0.0.0/0.0.0.0:51643 at (2016-09-25 12:49:56)
```

Ilustración 48. CoAPs – Client Hello con cookie.

- Paso 4 → una vez recibida la petición y aceptado el cliente como endpoint válido, el servidor procede a enviarle los siguientes datos los cuales podremos ver en la captura: ServerHelllo, Certificate, ServerKeyExchange, CertificateRequest y ServerHelloDone. Estos valores permitirán que el cliente reconozca al servidor, manteniéndolo sin estado, y guardar tanto su certificado como sus claves para utilizarlas posteriormente.

```
18 FINE [ClientHandshaker]: Processed SERVER_HELLO (2) message with
sequence no [1] from peer [localhost/127.0.0.1:5684] -
(org.eclipse.californium.scandium.dtls.ClientHandshaker.java:372)
doProcessMessage() in thread DTLS-Receiver-0.0.0.0/0.0.0.0:51643 at (2016-
09-25 12:49:56)

18 FINE [ClientHandshaker]: Processed CERTIFICATE (11) message with
sequence no [2] from peer [localhost/127.0.0.1:5684] -
(org.eclipse.californium.scandium.dtls.ClientHandshaker.java:372)
doProcessMessage() in thread DTLS-Receiver-0.0.0.0/0.0.0.0:51643 at (2016-
09-25 12:49:56)

18 FINE [ClientHandshaker]: Processed SERVER_KEY_EXCHANGE (12) message with
sequence no [3] from peer [localhost/127.0.0.1:5684] -
(org.eclipse.californium.scandium.dtls.ClientHandshaker.java:372)
doProcessMessage() in thread DTLS-Receiver-0.0.0.0/0.0.0.0:51643 at (2016-
09-25 12:49:56)

18 FINE [ClientHandshaker]: Processed CERTIFICATE_REQUEST (13) message with
sequence no [4] from peer [localhost/127.0.0.1:5684] -
(org.eclipse.californium.scandium.dtls.ClientHandshaker.java:372)
doProcessMessage() in thread DTLS-Receiver-0.0.0.0/0.0.0.0:51643 at (2016-
09-25 12:49:56)

18 FINE [ClientHandshaker]: Processed SERVER_HELLO_DONE (14) message with
sequence no [5] from peer [localhost/127.0.0.1:5684] -
(org.eclipse.californium.scandium.dtls.ClientHandshaker.java:372)
doProcessMessage() in thread DTLS-Receiver-0.0.0.0/0.0.0.0:51643 at (2016-
09-25 12:49:56)
```

Ilustración 49. Envío información Server a cliente.

- Paso 5 → en este paso el cliente envía al servidor los mismos datos sobre las claves que este le envió primero. Además le manda los *ciphersuits*.

```
18 FINE [ClientHandshaker]: Processing Change Cipher Spec (20) message from
peer [localhost/127.0.0.1:5684] -
(org.eclipse.californium.scandium.dtls.ClientHandshaker.java:179)
doProcessMessage() in thread DTLS-Receiver-0.0.0.0/0.0.0.0:51643 at (2016-
09-25 12:49:56)

18 FINE [ClientHandshaker]: Processed Change Cipher Spec (20) message from
peer [localhost/127.0.0.1:5684] -
(org.eclipse.californium.scandium.dtls.ClientHandshaker.java:372)
doProcessMessage() in thread DTLS-Receiver-0.0.0.0/0.0.0.0:51643 at (2016-
09-25 12:49:56)
```

Ilustración 50. Envío información cliente a server.

- Paso 6 → por último, el servidor envía sus *ciphersuits* al cliente y finaliza el *handshake*.

```
18 FINE [DTLSConnector$5]: Session with [localhost/127.0.0.1:5684]
established, now sending deferred message -
(org.eclipse.californium.scandium.DTLSConnector$5.java:837)
sessionEstablished() in thread DTLS-Receiver-0.0.0.0/0.0.0.0:51643 at
(2016-09-25 12:49:56)

18 FINE [Connection]: Session with [localhost/127.0.0.1:5684] has been
established - (org.eclipse.californium.scandium.dtls.Connection.java:837)
sessionEstablished() in thread DTLS-Receiver-0.0.0.0/0.0.0.0:51643 at
(2016-09-25 12:49:56)

18 FINE [ClientHandshaker]: Processed FINISHED (20) message with sequence
no [6] from peer [localhost/127.0.0.1:5684] -
(org.eclipse.californium.scandium.dtls.ClientHandshaker.java:372)
doProcessMessage() in thread DTLS-Receiver-0.0.0.0/0.0.0.0:51643 at (2016-
09-25 12:49:56)
```

Ilustración 51. Fin de handshake.

Una vez ya finalizado este primer paso, el cliente y el servidor ya pueden comenzar a intercambiar mensajes y esta vez de forma segura con DTLS.

A continuación podemos ver dos imágenes donde se muestran los resultados obtenidos por parte del cliente al realizar este primero una petición GET, pidiendo el valor *temperature*, y después, una petición PUT al servidor pidiéndole que modifique el valor temperatura por el enviado en el mensaje:

```
CLIENT: Temperature Sensor
---[Coap Request GET done]---

SERVER
==[ CoAP Response ]=============================================
MID     : 50447
Token   : 797926df97c69e3d
Type    : ACK
Status : 2.05
Options: {"Content-Format":"text/plain"}
Payload: 2 Bytes
----------------------------------------------------------------
12
================================================================
```

Ilustración 52. CoAPs - Petición GET.

```
CLIENT: Temperature Sensor
---Request PUT done with payload = 12---

SERVER
==[ CoAP Response ]=============================================
MID     : 15863
Token   : 32
Type    : ACK
Status  : 2.04
Options: {"Content-Format":"text/plain"}
Payload: 2 Bytes
----------------------------------------------------------------
12
================================================================
```

Ilustración 53. CoAPs - Petición PUT.

OBSERVABLE RESOURCES

El paquete **californium.observable.example** es el encargado de simular un entorno CoAP en el que se explota y prueba la característica *Observable*.

Tras ejecutar la clase Main.java de este paquete conseguimos poner en marcha la simulación. Hay que recordar que esta simulación consistía en crear un ejemplo formado por un cliente (sensor de temperatura) y un servidor, los cuales intercambiaban mensajes de tal forma que el cliente se subscribe a un *Observable Resource* del servidor y recibe periódicamente el valor de dicho recurso.

En la siguiente imagen se pueden ver los primeros datos que se obtienen en consola al simular este primer paquete ejecutando la clase *main*:

```
sep 24, 2016 9:01:49 PM org.eclipse.californium.core.network.config.NetworkConfig createStandardWithFile
INFORMACIÓN: Loading standard properties from file Californium.properties
sep 24, 2016 9:01:49 PM org.eclipse.californium.core.CoapServer start
INFORMACIÓN: Starting server
sep 24, 2016 9:01:49 PM org.eclipse.californium.core.CoapServer start
INFORMACIÓN: No endpoints have been defined for server, setting up server endpoint on default port 5683
sep 24, 2016 9:01:49 PM org.eclipse.californium.core.network.CoapEndpoint start
INFORMACIÓN: Starting endpoint at 0.0.0.0/0.0.0.0:5683
sep 24, 2016 9:01:49 PM org.eclipse.californium.core.network.CoapEndpoint start
INFORMACIÓN: Starting endpoint at 0.0.0.0/0.0.0.0:0
sep 24, 2016 9:01:49 PM org.eclipse.californium.core.network.EndpointManager createDefaultEndpoint
INFORMACIÓN: Created implicit default endpoint 0.0.0.0/0.0.0.0:56699
sep 24, 2016 9:01:49 PM org.eclipse.californium.core.CoapResource addObserveRelation
INFORMACIÓN: Successfully established observe relation between /127.0.0.1:56699#a223a6 and resource /Temperature
```

Ilustración 54. Observable - Captura consola.

En esta captura de pantalla podemos ver como lo primero que se hace es arrancar el servidor CoAP y el cliente, poniéndose este último a escuchar en la dirección IP 0.0.0.0 y el puerto 5683.

Como se puede ver en la última línea de la captura, se indica que el cliente ha conseguido establecer relación de forma correcta con el recurso “Temperature” alojado en el servidor (127.0.0.1).

Una vez arrancado el servidor, ya estará operativo para recibir cualquier petición por parte de clientes o *endpoints*.

Como primer paso para la subscripción a un recurso observable, el cliente deberá enviar un mensaje GET a dicho servidor con el valor *Observe* igual a 0.

A continuación podemos ver una imagen donde se muestran los resultados obtenidos por parte del cliente al realizar esta petición:

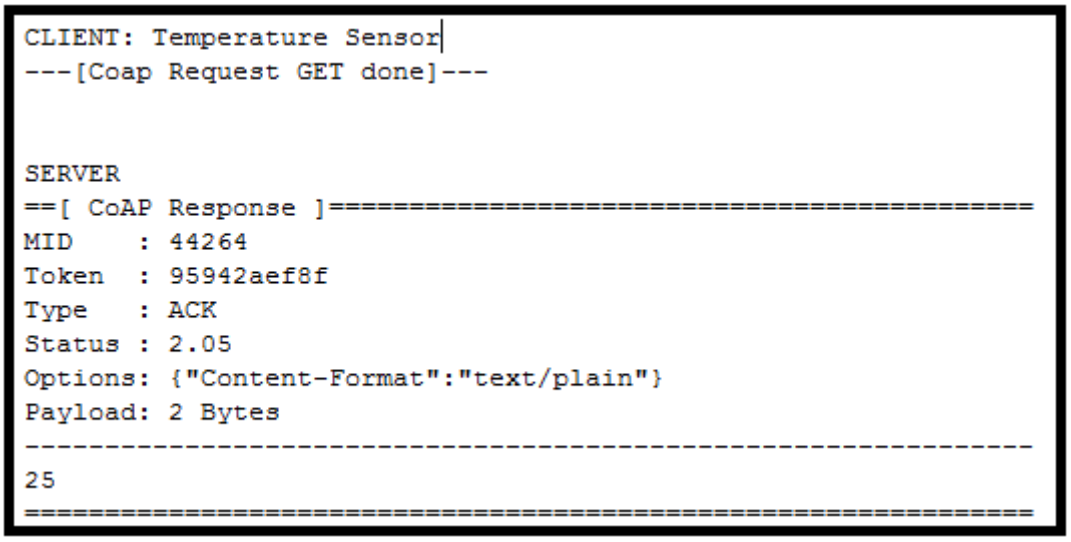

Ilustración 55. Observable - Petición GET.

Se puede observar como el sensor realiza la petición y obtiene como respuesta una trama bien definida con el valor 2.05 como estado, lo que significa que está de acuerdo. Además devuelve el valor de temperatura que tenía almacenado.

Una vez devuelto el ACK correspondiente al mensaje GET, el servidor comenzará a enviar notificaciones del valor del recurso (en este caso, *temperature*) al cliente. Estas notificaciones se enviarán periódicamente siguiendo un tiempo entre una y otra instanciado en el servidor.

```
==[ CoAP Response ]=========================================
MID     : 44263
Token   : a223a6
Type    : ACK
Status : 2.05
Options: {"Observe":1, "Content-Format":"text/plain"}
Payload: 2 Bytes
------------------------------------------------------------
25
============================================================

SERVER
==[ CoAP Response ]=========================================
MID     : 9396
Token   : a223a6
Type    : CON
Status : 2.05
Options: {"Observe":2, "Content-Format":"text/plain"}
Payload: 2 Bytes
------------------------------------------------------------
26
============================================================
```

Ilustración 56. Observable - Envío notificaciones.

En la imagen de arriba se puede ver dos notificaciones en las que se observa que el campo *Observe* va aumentando a medida que éstas se van enviando. También podemos ver como el valor de la temperatura varía.

5 CONCLUSIÓN Y LÍNEAS FUTURAS

5.1 CONCLUSIÓN

El objetivo de este Trabajo de Fin de Grado era realizar varias simulaciones de diferentes entornos explotando las características más importantes que tiene CoAP. Una vez creados estos entornos se evaluaría y estudiaría los resultados obtenidos.

La motivación del trabajo vino debido a que durante los últimos años se ha ido viendo como cada año aumentaba de forma exponencial el número de dispositivos conectados a Internet, previendo así un futuro en el que todas las cosas que nos rodeen estén conectadas entre sí. De ahí nace el concepto de Internet of Things, el cual es el motivo de este trabajo.

Se buscó un protocolo capaz de facilitar este intercambio de mensajes entre dispositivos de la vida cotidiana que por lo general no contarían con una gran cantidad de energía ni memoria. De esta forma se dio con CoAP, un nuevo protocolo especializado en pequeños dispositivos capaz de interconectar dos extremos.

Al ir a crear los entornos para explotar las características de CoAP había que elegir un software de desarrollo y tras analizar varios se eligió Californium. Este software, mediante sus librerías, nos daba las facilidades necesarias para realizar el trabajo y analizar de forma correcta las propiedades de CoAP.

Una vez elegidas las tecnologías, se al comenzar con el diseño y el desarrollo de los entornos fue cuando se dio la mayor dificultad del proyecto. Al ser un protocolo bastante nuevo y de momento no muy utilizado no existía una gran cantidad de información, sobretodo práctica, que pudiera ayudar en la implementación del código.

Finalmente se consiguió crear los 3 entornos del proyecto; el primero encargado de hacer una simulación de dos extremos con CoAP, el segundo con CoAP y DTLS y el tercero explotando la característica Observable Resource de CoAP.

Una vez creados los entornos se procedió a su análisis y se vio que estos funcionaban según lo esperado y cumplían con los objetivos planteados al comienzo del TFG. Por esto se puede decir que el trabajo ha sido exitoso ya que se ha conseguido simular un entorno del Internet de las Cosas y analizar su funcionamiento con el protocolo CoAP.

Además se ha visto que CoAP es un protocolo claramente útil para este tipo de entornos y que seguramente en algunos años será uno de los más utilizados.

5.2 LÍNEAS FUTURAS

En cuanto a las futuras líneas de trabajo sobre este proyecto cabe descartar que se dispone de un gran margen de ampliación. En este apartado se proponen diferentes opciones que podrían mejorar el escenario planteado.

Una futura línea de trabajo podría ser realizar crear un entorno real completo formado por varios dispositivos como sensores, bombillas e incluso smartphones. Creando esta aplicación podría estudiarse el comportamiento de este protocolo con factores externos como el ruido, algo que en un entorno simulado no puede probarse.

Otra posible mejora podría ser la creación del mismo escenario de simulación pero esta vez con otro protocolo parecido como MQTT o HTTP. De esta forma se podría realizar una comparación directa y exacta del comportamiento de estos protocolos así como de la velocidad, gasto de energía, tamaño de código,…

A su vez también se podría diseñar el mismo entorno pero esta vez utilizando otro lenguaje de programación como C o C#. Con esto podríamos comprobar cuál de los dos funciona mejor. Se podría analizar la cantidad de código, la complejidad, como trabaja el software proporcionado,…

También se podría crear un nuevo entorno en el que se simulara un ejemplo de CoAP multicas. De esta forma podríamos ver cómo funciona.

6 APÉNDICE: ORGANIZACIÓN

6.1 MARCO REGULADOR

En cuanto al marco regulatorio relacionado con el Internet de las Cosas no hay una gran cantidad de normativas debido a que de momento es más una idea de futuro, aunque no muy lejano, a un hecho del presente.

En regulación para el IoT hay que tener en cuenta tanto los dispositivos como las redes y su seguridad, teniendo principalmente en cuenta los datos que se intercambian. Algunas de las materias que se deben destacar en estos campos y que principalmente vienen reguladas por la Unión Europea son: [16]

- En cuanto a la **estandarización**, la **Directiva 2014/53/UE**, la cual habla sobre la armonización en la comercialización de aparatos radioeléctricos de los Estados miembros, siendo esta fundamental para el desarrollo conjunto y armonizado de la tecnología.
- Respecto a la **privacidad, protección de datos y propiedad**, el nuevo Reglamento General de Protección de Datos (cuyas siglas en inglés son GDPR) será a partir de 2018 la normativa a seguir por la UE en cuanto al tratamiento de datos personales. Hasta entonces sigue en vigor la Directiva sobre Protección de Datos de 1995.
- Por otra parte, la **ciberseguridad** se trata en la Directiva de Seguridad en Redes y Sistemas de la Información (en inglés NIS). Su objetivo es instaurar un nivel común en toda la UE y mejorar la coordinación de los estados que la forman.

6.2 PLANIFICACIÓN DEL PROYECTO

Para la planificación del proyecto, se ha construido el siguiente Diagrama de Gantt realizado con el programa Gantt Project.

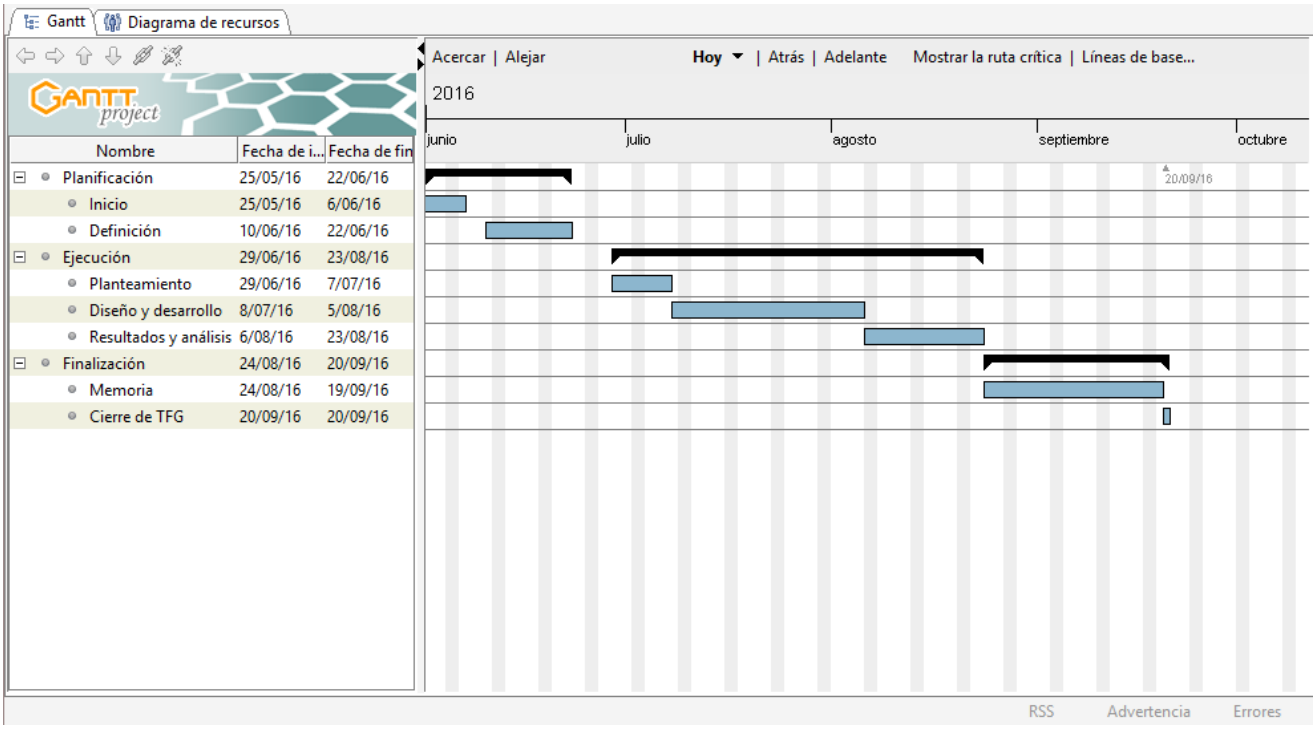

Ilustración 57. Diagrama de Grantt del proyecto.

Como se puede observar en el diagrama, la fecha de inicio del TFG fue el XX de YY de 2016 y se comenzó con la etapa de Planificación, la cual consiste en lo siguiente:

Planificación: Desde el día 25 de mayo de 2016 hasta el 28 de junio de 2016.

- Inicio → Comienza el día 25 de mayo y acaba el 10 de junio.
    - Búsqueda de un problema.
    - Análisis del entorno que se quiere estudiar.
    - Análisis de estudios previos sobre el tema.
- Definición → Comienza el día 11 de junio y acaba el 28 de junio.
    - Presentación de soluciones al tutor.
    - Establecimiento de los objetivos.

La segunda etapa del Proyecto se trata de la **Ejecución**. Es la más larga ya que comprende desde el planteamiento hasta la evaluación de los resultados. Está compuesta por las siguientes partes:

Ejecución: Desde el día 29 de junio de 2016 hasta el 23 de agosto de 2016.

- Planteamiento → Comienza el día 29 de junio y acaba el 7 de julio.
    - Preparación del entorno de trabajo.
    - Estudio de las librerías a utilizar.
    - Estudio del SDK de Java.

- Diseño y desarrollo → Comienza el día 8 de julio y acaba el 5 de agosto.
    - Diseño de los diferentes entornos a simular.
    - Desarrollo de la solución.
- Resultados y análisis → Comienza el día 6 de agosto y acaba el 23 de agosto.
    - Realización de diferentes pruebas.
    - Revisión del correcto funcionamiento de los entornos.
    - Análisis de la tecnología utilizada.
    - Presentación de conclusiones.
    - Planteamiento de posibles mejoras en el sistema.

La última etapa es la **Finalización** que consiste en la realización y entrega de la Memoria. Las partes por las que está formada son:

Finalización: Desde el día 24 de agosto de 2016 hasta el 20 de septiembre de 2016.

- Memoria → Comienza el día 24 de agosto y acaba el 19 de septiembre.
    - Realización de la Memoria.
    - Realización del anexo en inglés.
    - Revisión del trabajo por parte del tutor.
    - Corrección de los errores.
- Cierre del Trabajo de Fin de Grado → Se produce el día 20 de septiembre de 2016.

6.3 PRESUPUESTO

En este apartado se realizará un desglose general de todos los costes que produce realizar este proyecto. Para ello, nos basaremos en las diferentes tareas del proyecto anteriormente mencionadas.

TAREA	INTERVALO DE TIEMPO
Inicio	17 días
Definición	19 días
Planteamiento	9 días
Diseño y desarrollo	29 días
Resultados y análisis	18 días
Finalización	31 días

Tabla 4. Intervalo de tiempo para las tareas.

Para calcular el número de horas de trabajo que se han invertido en cada una de las tareas se tendrán en cuanto tanto los días lectivos como los festivos. Si se desean más detalles, leer el apartado anterior.

TAREA	HORAS DEDICADAS
Inicio	17 días x 3h/día
Definición	19 días x 3h/día
Planteamiento	9 días x 3h/día
Diseño y desarrollo	29 días x 4h/día
Evaluación y resultados	18 días x 4h/día
Finalización	31 días x 4h/día
TOTAL	123 días = **447 horas**

Tabla 5. Horas dedicadas al TFG.

En total se han destinado 447 horas de trabajo a la realización de este TFG.

▪ COSTES DEL MATERIAL

Para llevar a cabo el cálculo de los **costes materiales**, se tiene en cuenta tanto un ordenador con capacidad y potencia suficientes para soportar una máquina virtual, las licencias correspondientes al software y el dispositivo de seguimiento de ojos de la empresa "*The Eye Tribe*" como todos los recursos necesarios para la ejecución del Proyecto tales como electricidad para el funcionamiento de los equipos, como conexión a Internet y páginas relacionadas para realizar la tarea de estudio e investigación. Por todo ello, se tienen los conceptos descritos en la siguiente tabla:

CONCEPTO	Coste	Dedicación (meses)	Vida útil (meses)	COSTE imputable
Ordenador x64 – Intel Core i7 – CPU 3.60GHz	900 €	4	38	94 €
Licencia Windows 8.1 Pro	100 €	4	36	11 €
Licencia Microsoft Office 2015	100 €	3	12	25 €
Gastos de electricidad	70 €	-	-	70 €
Conexión a Internet – 300Mbps	100 €	-	-	100 €
COSTE TOTAL DEL MATERIAL				**300 €**

Tabla 6. Coste del material.

▪ COSTES DEL PERSONAL

Para calcular los costes de personal del proyecto se va a tener en cuenta a las personas que han participado en él, el tutor y el alumno.
Con el fin de describir estos datos, a continuación se muestra una tabla basada en la categoría, el tiempo invertido en la realización (en horas) y el sueldo (en Euros) de la persona.

CATEGORÍA	COSTE (€/h)	TIEMPO (h)	TOTAL
Dr. Ingeniero (Tutor)	60	30	**1.800 €**
Alumno	30	447	**13.410 €**
COSTE TOTAL DEL PERSONAL			**15.210 €**

Tabla 7. Coste personal.

▪ COSTE TOTAL DEL PROYECTO

Finalmente, se lleva a cabo una tabla que representa el **coste total** del Proyecto con la suma de los costes materiales, personales e impuestos.

CONCEPTO	PRECIO
Costes del material	**300 €**
Costes del personal	**15.210 €**
COSTE TOTAL DEL PROYECTO	**15.510 €**

Tabla 8. Coste total del proyecto.

El coste total de la realización total del Proyecto estudiado es de **15.510** €.

ENLACES, REFERENCIAS Y BIBLIOGRAFÍA

[1] JULIANNE TWINING, "*BEHIND THE NUMBERS: GROWTH IN THE INTERNET OF THINGS*", 2015. Disponible online: https://www.ncta.com/platform/broadband-internet/behind-the-numbers-growth-in-the-internet-of-things-2/ (Último acceso: agosto 2016)

[2] ALEJANDRO SÁNCHEZ, "*IoT – Internet of Things*", 2016. Disponible online: http://www.mejor-antivirus.es/terminologia-informatica/iot-internet-of-things.html (Último acceso: agosto 2016)

[3] Waldner, Jean-Baptiste. Inventer l'Ordinateur du XXIeme Siècle. London: Hermes Science. 2007. pp. p254. ISBN 2746215160. (Último acceso: agosto 2016)

[4] Z. Shelby, ARM, K. Hartke, C. Bormann. "*The Constrained Application Protocol (CoAP)*", 2014. Disponible online: https://tools.ietf.org/html/rfc7252 (Último acceso: agosto 2016)

[5] Octavio Torres. "*AMON Formato de datos para intercambio de información M2M*", 2013. Disponible online: https://unpocodejava.wordpress.com/category/m2m/ (Último acceso: agosto 2016)

[6] Zach Shelby. "*ARM IoT Tutorial*", 2014. Disponible online: http://www.slideshare.net/zdshelby/coap-tutorial (Último acceso: agosto 2016)

[7] Kevin Ashton, "*That 'Internet of Things' thing*", 2009. (Último acceso: agosto 2016)

[8] Gartner. "*Gartner Says the Internet of Things Installed Base Will Grow to 26 Billion Units By 2020*", 2013.

[9] Mathieu Paumard. "*Qué es el Internet of Things y cómo cambiará nuestra vida en el futuro*", 2011.

[10] Nicolás Rivera. "*AMON Formato de datos para intercambio de información M2M*", 2015. Disponible online: https://hipertextual.com/2015/06/internet-of-things (Último acceso: agosto 2016)

[11] Nicolás Rivera. "*AMON Formato de datos para intercambio de información M2M*", 2015. Disponible online: https://hipertextual.com/2015/06/internet-of-things (Último acceso: agosto 2016)

[12] Matthias Kovatsch, Julien Vermillard. "*Hands-on with CoAP*", 2014. Disponible online: https://docs.google.com/presentation/d/1dDZ7VTdjBZxnqcIt6qoX742d6dHbzap-D_H8Frf3LRE/present?pli=1&ueb=true&slide=id.g3740c9df4_5_71 (Último acceso: agosto 2016)

[13] Zach Shelby. "*ARM IoT Tutorial*", 2014. Disponible online: http://www.slideshare.net/zdshelby/coap-tutorial (Último acceso: agosto 2016)

[14] David Gallardo. "*Iniciándose en la plataforma Eclipse*", 2012. Disponible online: https://www.ibm.com/developerworks/ssa/library/os-ecov/ (Último acceso: agosto 2016)

[15] Apache Maven Project. "*Maven*", 2016. Disponible online: https://maven.apache.org/what-is-maven.html (Último acceso: agosto 2016)

[16] Eclipse. "*MQTT and CoAP, IoT Protocols*", 2014. Disponible online: https://eclipse.org/community/eclipse_newsletter/2014/february/article2.php (Último acceso: agosto 2016)

[17] George Kontopoulos. "*Resource constrained protocols for IoT: 6LoWPAN, MQTT & CoAP*", 2016. Disponible online: https://www.linkedin.com/pulse/resource-constrained-protocols-iot-6lowpan-mqtt-coap-kontopoulos (Último acceso: agosto 2016)

[18]Xi Chen. "*Constrained Application Protocol for Internet of Things*", 2016. Disponible online: http://www.cse.wustl.edu/~jain/cse574-14/ftp/coap/ (Último acceso: agosto 2016)

*[19]*S. Raza, H. Shafagh, etc. "*Lithe: Lightweight Secure CoAP for the Internet of Things*", 2016. Disponible online: http://web.cs.wpi.edu/~rek/IoT/Lithe_F15.pdf (Último acceso: agosto 2016)